Sheffield Hallam University

Incremental Perception in Swarm Robotics

Mir Immad ud din

M.Sc. Computer and Network Engineering
Full-Time
2005-06

First supervisor: Dr. Bala P. Amavasai
Second supervisor: Dr. Stuart Meikle

This thesis is submitted in partial fulfilment
of the requirements for the degree of
Masters of Science
in Computer and Network Engineering

I said to myself, I have things in my head that are not like what anyone has taught me - shapes and ideas so near to me - so natural to my way of being and thinking that it hasn't occurred to me to put them down. I decided to start anew, to strip away what I had been taught.

(Georgia O'Keeffe; 1887 - 1986)

Incremental Perception in Swarm Robotics

by

Mir Immad ud din

Submitted to the Faculty of ACES on September 25, 2006 in Fulfillment of Requirements for the Degree of Master of Science in Computer and Network Engineering

Abstract

Incremental Perception is a novel term in Swarm Robotics, which opens a whole world for thought and introduces many areas of research. Theme of the present work is to adapt a Hybrid (decentralised and centralised) approach to *Incremental* Object Recognition by un-intelligent Robots equipped with very few sensors (e.g. only touch sensors). These Robots exhibit Behaviour-Based Cooperation that builds upon simple rules in absence of any central controller, thus conforming to a Decentralized Model. Interaction via Sensing approach is used with no communication between Robots at all. This makes the present work unique, as the Robots not only sense the presence of an object they go ahead by sharing it with other Robots merely relying upon their behaviour, hence the term *Incremental* Perception/Object Recognition.

The work also addresses Formation and Marching problem for which a Centralized approach has been adapted. Once Robots are ready to transport the object they have found, their behaviour turns the model into Centralized Architecture by selecting a 'Leader', who would guide the swarm back home.

Modelling language Net Logo has been used to simulate and test Algorithms. The present thesis also presents a literature survey of related work and identifies future areas with research potential.

Project Supervisor
Dr. Bala P. Amavasai
Microsystems and Machine Vision Laboratory
Faculty of ACES

Co-Supervisor
Dr. Stuart Meikle
Nokia Research

Acknowledgements

I owe my success to having listened respectfully to the very best advice, and then going away and doing the exact opposite. G. K. Chesterton (1874 - 1936) [1]

All the way through our lives, we exhibit Swarm Behaviour and hardly ever notice it. The fact that I've come so far as completing my thesis for a Masters degree, is a result of behaviours of members of swarm that have been around me who have behaved in a way so as to support me. I am thankful to Almighty Allah for his blessings and giving me the strength to overcome all hardships in the way, for receiving an honour that a feeble percentage of world's population is able to achieve.

This work would not have been possible without the never-ending support and patience of Dr. Bala Amavasai who has not only been my supervisor, but also a source of inspiration with endless ideas and someone who always had a way to do things in a better way. I express my heartfelt gratitude for all his encouragement and challenging my thoughts, guiding my mind to work in an organized way and for giving me an entirely new professional approach.

I am thankful to the Faculty of ACES and the Sheffield Hallam University for all the help and support that I have been receiving throughout the academic year. The knowledge and experience that I gained during my stay is priceless, and is my most precious asset. I have to accept that it is here that I learned what hard work really is and how important dreams are.

I am highly indebted to all my teachers, for it is they who have made my what I am today. I am thankful to my mother, my first teacher; Mrs. Beryl B. Franklin, one of my junior school teachers; Mr. Chaurhary M. Amin, my physics teacher; Dr. Shahid H. Bokhari, Operating Systems teacher who has always been a source of inspiration for me; and Dr. Bala P. Amavasai for teaching me how to make dreams come true.

[1] well, just a quotation…..

This work has been possible because of unseen endless efforts by a lot of people around me. I dedicate my work to

My Father

for he didn't tell me how to live, but he lived it himself and let me watch him do it.

(Clarence Budington Kelland)

My Mother

for she dreamt great dreams for me, and then let me chase the dreams I had for myself and still loved me just the same.

(Author Unknown)

My Wife

for understanding my dreams and being my best friend, for standing beside me in rain and in sunshine, for filling my life with joy and loving me without end.

(Mark Twain)

My Teachers

Give me a fish and I eat for a day. Teach me to fish and I eat for a lifetime.

(Chinese Proverb)

Dr. Shahid Bokhari

Sometimes one man with courage is a majority.

(Andrew Jackson)

Contents

Chapter 5. Statistical Analysis and Results

Chapter 6. Improvements and Future Work

Appendix A: Code

References

Bibliography

Figures and Images

Tables

Graphs

Equations

Chapter 1.
Introduction

1.1 History

Swarm or *Swarm Behaviour* are terms used by people from a diverse domain of research interests, ranging from Biology, Military, Cooperative Robotics and Artificial Intelligence. Biologists use Swarm to define cooperative behaviour of insects and some other animals. This term has also been used by Military Historians to define battlefield tactics.

In an arena of cooperative Robotics, the term was introduced by Gerardo Beni (1988) to describe his work on Cellular Robotic Systems. It has since been used to describe a variety of behaviours and approaches in areas as diverse as Biology (cooperative behaviour of insects), Military Tactics, Robotics and Distributed Artificial Intelligence.

Many explanations have emerged for the term, each adapted by a different set of technologists working in different fields. Parunak (2003) states that Biologists use *Swarm* to define "decentralised self-organizing behaviour of (usually simple) animals" whereas Military Historians use it to describe "a battlefield tactic that uses decentralised pulsed attacks".

Hackwood and Beni (1991) classify *Swarm* intelligence as a property of system of non-intelligent Robots exhibiting collectively intelligent behaviour.

Bonabeau *et al.* (1999) have defined *Swarm* Robots as distributed problem solving devices inspired by collective behaviour of social insect colonies and other animal societies.

Whatever may the definition be, the basic idea in one way or the other is inspired by Decentralised Cooperative behaviour of social animal specie. This behaviour in general is simple (like in ants) but gives rise to complex patterns.

1.2 Inspiration

Earliest work on *Swarm*s comes from the research on Social Insects. Biologists have long been inspired by cooperative behaviours of insects like Ants and Bees, which led to intense research on their behaviours. It was realised that although the collective behaviour of these insects may seem very complex (Figure 1), yet was based upon very few simple rules.

The existence of Ants and Bees as some of the most successful species on the planet has probably been the strongest inspiration behind adapting a *Swarm* approach in Cooperative Robotics. The term '*Swarm'* was first used by Beni in his work on Cellular Robotic System and has since persisted to describe a Decentralised approach to Collaborative Robotic Applications.

Swarm and *Swarm* Intelligence are now being used in a variety of disciplines. Wedde and Farooq (2005) have applied a S*warm* approach in their Mobile ad-hock Network model, Cianci and Martinoli (2005) present their work on load balancing in sensor networks. Montemanni and Gambardella (2005) adapt a *Swarm* approach to solve connectivity

problem in wireless networks while Wan *et al.* (2005) have presented a flexible distributed network partitioning solution.

(a) (b)

Figure 1: (a) Honeybees forage near a sugar-water feeder doped with explosives stimulants, part of bee-training experiments at Sandia National Laboratories. (b) Ants surround a toxic gel mistaken that it is a food source. Toxic gels like this are an effective remedy from ants as it is fed to offspring as well, thus destroying the whole ant colony.

1.3 The Project

The present project establishes a technique called '*Incremental Perception*' for object recognition in a *Swarm* of Robots. Incremental perception is the information built-up and shared by S*warm* of simple Robots (equipped with merely touch sensors and no visual device) to collectively recognize an object as their target. There is no explicit communication between the Robots and the information being shared is only through display of behaviours.

1.3.1 The Swarm Behaviour

Each Robot displays a small set of simple yet consistent behaviours which gives rise to a complex colony. The agents wander around in the world looking for an object, unaware by the presence of other Robots. If however two Robots come close enough such that

there is a possibility of collision between them, they turn around and adapt a different route. The model behaves in a Decentralised fashion until the first Robot finds an object.

1.3.2 Object Recognition

Robots have a finite field of adjustable vision. At each step during their motion, they look around in a pre-defined way to check whether they can see any object. As soon as the first Robot finds an object, it becomes stationary and creates a Potential field around it. We will see in Chapter 3 that this Robot behaves as the leader. Any other Robots entering the Potential field recognize its presence and try to move in a direction downhill with respect to the field, as it is the direction in which the field originates from. While moving within the potential field, Robots maintain a factor of randomness that can be controlled externally.

1.3.3 Incremental Perception

Any member of the *Swarm* that locates an object in presence of a potential field will assume that it is the same object that one or more members have previously found, as a field can only exist after at least one *Swarm* member finds an object. It thus adds to the task by informing Master Controller about its coordinates as soon as it locates the object. Please note that the master controller has not played any role yet and the *Swarm* has behaved in a purely Decentralised fashion.

1.3.4 Centralised Approach

The *Swarm* can now behave in two ways. It can either wait for the master node to analyze the information from stationary Robots and extract the shape of object before taking any decision about moving the object back home. Second behaviour can be like African Ants, which try to bring back any object that they find during an attack. The present model has applied a hybrid approach; it started with a Decentralised architecture, and is ready to turn into Centralised as soon as the swarm finds the target object. Rules for this change have carefully been designed. Please note that this change is in terms of the way the *Swarm* behaves under special conditions outlined above.

1.4 Resources

This project is simulation based with a possibility of extension to a hardware application in the form of development of a new Robot, or testing of algorithms on available Robots like Khepera (Figure 2).

Figure 2: Khepera II Robot; is a 70 x 30 mm, 80g Robot with 8 IR sensors and 512 KB RAM.

There were many options available for the choice of simulation language with the possibility of developing the model in a general purpose programming language like C or Java. However, in the later case, most of the efforts would have been spent in details of developing the modelling environment i.e. the agents, world and rules for their interoperability. Thus choice had to be made between one of modelling languages available, namely *breve* and NetLogo.

1.4.1 Breve

The *Breve* project started at Hampshire College as a thesis by Jon Klein, however major developments in the modelling environment came during his stay at Chalmers University. The system has experienced vast acceptance by researchers working on Genetic Algorithms and Evolutionary Models.

Breve allows simulation of evolving multiple agents for whom user defined behaviours govern their life in a 3D environment.

It is a free simulation environment designed for multi-agent simulations that allows users to define the behaviours of autonomous agents in a continuous 3D world, and then observe how they interact. It also supports a visualisation engine, possibility to integrate physical laws into the simulation and uses easy-to-use scripting languages. (Klein J, 2003)

The world in *breve* is a 3D space, hence facilitating 3D spatial simulations. The agents can occupy this 3D space and use it to move about and show predefined behaviours. These agents can also be made to comply with simple physical laws, hence making simulations more close to a real world. This 3D spatial environment makes *breve* different from many simulation languages that offer a 2D environment for their agents. (Klein, J. 2003, breve Documentation: version 2.4; Section 2.1.2)

By enabling the feature of Physical Laws into a simulation, breve agents can be forced to behave like real life objects in a real world. For example, a ball may be placed in the air and physical simulation can be used to make the ball realistically fall toward the floor and bounce. Among other things, physical simulation can be used for realistic simulation of Robots, vehicles and animals.

This ODE Physical Simulation Engine (Figure 3) makes breve a strong candidate for simulating real life phenomenon by allowing programmers to test their models and see how they would behave in real life. However the documentation suggests that enabling the feature may make the model considerably complex, and devotes Section 7.1 to discuss scenarios where the use of such a model would be appropriate.

1.4.2 Languages in Breve

The steve Language

The *breve* simulations are normally written in *'steve',* an object-oriented language that allows programmers to quickly structure sophisticated simulations while shunning overheads of programming in a general purpose programming language. The *steve* objects can either appear in the simulated world, or they can be abstract and used for data storage or any other purpose other than appearance in the simulated world. The language is described chapter 3 of *breve* documentation. (Klein J. 2003, Documentation: version 2.4)

Push

The *Push* programming language was developed specifically for genetic and evolutionary computation. Among its qualities for such applications is its combination of an unusually simple syntax with the ability to work flexibly with multiple datatypes. (Spector *et al*.2005)

Genetic programming in *breve* is now done with the aid of *Push*, which has an additional feature of being an evolvable language.

Programs written in *Push* are evolved with the aid of a system called the PushGP. It has been used for a variety of applications, ranging from intelligent agent design to automatic quantum computer programming. Features include:

- Multiple data types without constraints on code generation or manipulation.
- Arbitrary modularity without constraints on code generation or manipulation.
- Evolved module architecture with no extra machinery.
- Support for explicit, arbitrary recursion.
- Support for ontogenetic "development" of code as it runs, via code-manipulation instructions.

The Push3 EXEC stack supports powerful and parsimonious control regimes through explicit manipulation of the stack of expressions that are queued for execution. These control regimes include standard iteration, several forms of recursion based on code manipulation, combinators, and named subroutines, and less conventional strategies that are difficult to classify. A straightforward genetic programming system that produces Push3 programs (PushGP) can routinely produce solutions that incorporate a range of these control regimes; examples were provided here for reversing and sorting lists and for computing factorials, Fibonacci numbers, powers of 2, and parity. Application of these techniques to real-world problems is currently in progress. (Spector *et al*.2005)

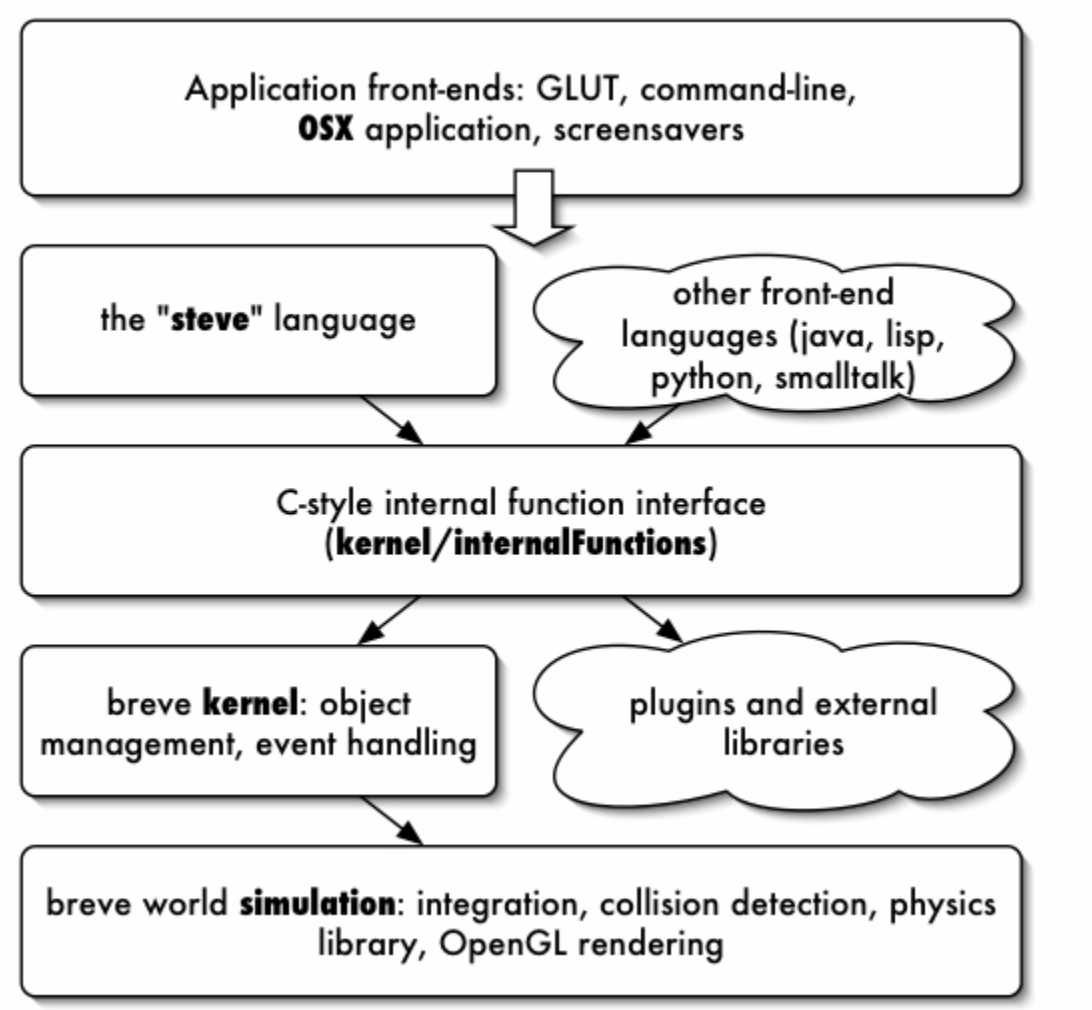


Figure 3: The *breve* software architecture; Image source: Breve Documentation: version 2.4 Chapter 14: 'The *breve* Source Code'

1.4.3 NetLogo

Currently under development at CCL (Centre for Connected Learning and Computer-Based Modelling), NetLogo is a cross-platform multi-agent programmable modelling environment developed by Uri Wilensky in 1999. It is developed on Java platform (version 1.4.1) and thus inherits many features of the parent language like portability, garbage collection and the strict math library. It is released under a license that appears to be much liberal by allowing unrestricted use (including commercial use) but there are some restrictions on redistribution and/or modification. (Wilensky 1999)

Wilensky (1999) states that NetLogo offers deterministic scheduling algorithms and uses Java's strict math library, both features make it an ideal choice for modelling of real time phenomenon. Being Java based, and running on a JVM instead of underlying operating system, NetLogo simulations give identical results irrespective of underlying operating system and hardware. It has widely been used to simulate natural phenomenon and real life scenarios, some of which appear in the models library.

Although it works in the fashion of run time interpreted languages, but does include a compiler. The compiler does not produce native code or Java byte code rather it produces a custom intermediate representation that can be interpreted more efficiently than the original code. The NetLogo team claims that they are working on a compiler that would generate a byte code for Java’s just in time compiler. (Wilensky 1999)

A NetLogo installation can be run from a read-only file system like a CD-ROM or a DVD-ROM. The modelling environment has widely been used to simulate real life scenarios, cooperative and eusocial agent colonies and foe analysing communal behaviours.

1.4.4 NetLogo's Strengths

Java

Having an underlying Java platform itself brings many useful features, namely Platform Independence, Architecture Independence, Garbage Collection, and possibility to use strict math Library.

BehaviourSpace

Integration of BehaviourSpace is one of the features that make NetLogo an ideally convenient option for simulation. It is a software tool that allows running a single model over and over again with different system parameters. This along with file I/O makes is exceptionally easy to output data from a model for statistical analysis and lets the programmer explore the model's space of possible behaviours. (Wilensky 1999a)

The GoGo Board

GoGo Board is an externally serially interfacable hardware that contains sensors, motors, light bulbs, LEDs, relays and certain other devices. It is an open source general purpose board designed by Arnan Sipitakiat at the MIT Media Lab. The GoGo Board has 8 sensor ports and 4 output ports, and also a connector for add-on boards. Using the GoGo Board extension, NetLogo models can interact with the physical world in two ways. First, it can gather data from the environment, such as temperature, ambient light, or user input. This information can be used by the model to change or calibrate its behaviour. Secondly, it can control output devices - NetLogo could control motors, toys, remote controlled cars, electrical appliances, light bulbs, and automated laboratory equipment. (Wilensky 1999a)

Shapes Editor

The Shapes Editor allows creating and saving turtle designs. NetLogo uses fully scaleable and rotatable vector shapes, which means it lets you create designs by combining basic geometric elements, which can appear on-screen in any size or orientation. (Wilensky 1999a)

System Dynamics Modeller

System Dynamics is a concept different from the normal agent-based approach and allows the designer to understand how things relate to one another. An agent-based model allows the designer to set the behaviour of agents and to define the behaviours of populations of agents. (Wilensky 1999a)

HubNet Architecture

HubNet is a technology that allows NetLogo to run client-server based distributed and interactive simulations. The activity leader uses the NetLogo application to run a HubNet activity.

There are two types of HubNet available. With Computer HubNet, participants run the HubNet Client application on computers connected by a regular computer network. In Calculator HubNet, created in conjunction with Texas Instruments, participants use TI-83+ graphing calculators as clients that communicate via the TI-Navigator system. (Wilensky 1999a, c)

1.4.5 Limitations of NetLogo

- Integers in NetLogo must lie in the range –2147483648 to 2147483647; this range if exceeded gives incorrect results instead of an error.

- The uphill and downhill reporters sometimes return incorrect answers for turtles which are standing on patch boundaries; NetLogo team is already working to fix it and recommends `uphill4` and `downhill4` primitives instead.

- The 3D View doesn't work on some graphics configurations; on others the 3D View works but 3D full screen mode doesn't.

1.5 Why NetLogo

Tobias and Hofmann (2004) suggest that Breve is not suitable for simulating social behaviour in groups of multiple agents, while NetLogo is classified as well suited for modelling complex systems developing over time. (Pozdnyakov 2006)

NetLogo basically is a programmable modelling environment for simulating interactive social and natural phenomenon, which can theoretically accommodate heavily populated models of up to thousands of agents, a feature that makes it ideal for Swarm simulations.

License under which NetLogo is released is liberal and allows unrestricted use. The free modelling environment can even be used for commercial purposes.

Deterministic Scheduling Algorithms have been used that make it an ideal choice for modelling of real time phenomenon.

NetLogo simulations give identical results irrespective of underlying operating system and hardware. This along with the fact that NetLogo models can be run as applets, allows the ease of demonstration to a wide audience while producing the same results.

Summary

Cooperative/Swarm Robotics are areas that have heavily been inspired by behaviours of eusocial insect colonies. The present work presents a decentralised model that builds upon low-level behaviours of Homogeneous Robots. Robots move around in the world, aware of their own location but unaware of the presence of other Robots. It is only when two Robots come very close to each other that they realize the presence of another agent.

The S*warm* has a clearly defined task, finding an object and after recognizing it as its target, try to bring it back. The task is accomplished without the aid of any visual device.

The model is simulated in NetLogo, which is one of the modelling languages of StarLogo series.

Chapter 2.
Theory and Related Work

Swarm Intelligence is described as "to design a system that while composed of un-intelligent units, is capable, as a group to perform tasks requiring intelligence, the so-called Swarm intelligence". (Beni and Wang 1989)

Most of the work found on Object Recognition in Robotics/Multi Robotics involves the use of a visual device (usually a camera) and high level image processing algorithms that require intense computing resources (Tuytelaars et al. 2000). The work presented in this thesis is unique in the context that it uses un-intelligent Robots with sensors as simple as touch sensors. The object recognition, here, is a result of cooperative behaviours exhibited by the swarm without using any sophisticated image processing technique. The movement of Robots themselves helps to extract the shape of object / target. This chapter briefly discusses the Swarm Robotics in general, and topics like Behaviour Based, Centralised / Decentralised Architectures in particular, and lets the reader appreciate the novelty of technique described in Chapters 3 and 4.

2.1 Early work on *Swarm* Robotics / Cooperative Robotics

Swarm Robotics evolved from Cooperative Multi-Robotic Systems, in fact the term Swarm was first used by Beni in his work on Cellular Robotic Systems (Beni 1998). The study of Cooperative Robotics dates back to 1940s when Walter et al. started exploring

the cooperative behaviour of turtle-like Robots equipped with touch and light sensors (cited in Wiley-Interscience 1990).

Reynolds (1993) evolved the control system of a group of Robots placed in an environment with static obstacles and a manually programmed predator for the ability to avoid obstacles and predation. Despite the results described in the paper are rather preliminary, some evidences indicate that coordinated motion strategies began to emerge. (Reynolds 1993 cited by Theraulaz et al. 1999)

Baldassarre (2002) evolved a group of Robots to aggregate and move together towards a light source. He says, "The main advantage of evolution of group of Robots is that it is an ideal framework for synthesizing Robots whose behaviour emerge from a large number of interactions among their constituent parts."

Theraulaz and Bonabeau (1995) evolved a population of constructor agents who collectively build a nest structure by depositing bricks according to their perception of the local environment and to a set of behavioural rules.

Theraulaz et al. (1999) attempted to design algorithms for distributed problem-solving devices inspired by the collective behaviour of social insect colonies, such as ants.

Martinoli (1999) used artificial evolution to synthesize the control system of a group of simulated Khapera Robots (Mondada et al. 1993) that were asked to find food items randomly distributed on an arena.

In the attempt to study the evolutionary origin of herding, Werner and Dyer (1993) co-evolved two populations of predators and prey creatures that were selected for the ability to catch prey and to find food and escape predators respectively. (Werner and Dyer 1993 cited by Theraulaz et al. 1999)

Ward et al. (2001) evolved groups of artificial fish able to display schooling behaviours.

2.1.1 Cellular Robotic Systems

Work on Cellular Robotic System laid the foundation of Swarm Robotics. The cellular Robotic system of Beni (1989) has many agents that behave in a way so as to produce pattern generation and self-organization. Self-organization was defined as a property of the agents of a system to distribute themselves optimally for a given task.

2.1.2 The CEBOT

Cellular Robots (CEBOT) is a Dynamically Reconfigurable, Fault Tolerant, Self-organizing, Self Evolving system. Designers of the system have given it a wide range of properties to produce a multiple Robotic system with distributed autonomous Robots who are able to show swarm/collective behaviours. CEBOT consists of many basic functional units and has been used to experiment with intelligent control and system architecture. CEBOT has been recommended for a variety of applications like space applications, intelligent transportation systems and intelligent manufacturing systems etc. Owing to Micro Manufacturing Technology and Nano Electro-Mechanical Systems (NEMS), CEBOT can be extended to micro-robotic applications. (Fukuda 2004)

2.1.3 Alliance / L-Alliance

Developed by Parker (1994a) as his PhD. project, Alliance consisted of Independent Heterogeneous agents. It is a behaviour-based model in which communication between agents enables them to (limitedly) interpret the affects of behaviour of other agents. In his PhD Thesis (Parker 1994b) he presents L-Alliance, an extension to the original Alliance model, which uses *reinforcement learning* to adjust the parameters controlling behaviour set activation.

Alliance/L-Alliance model has widely been used to analyze several Marching and Formation problems, both in simulation and on Hardware.

2.2 Centralised Model

Yao et al. (1997) say that the most fundamental decision to be made while defining group architecture for a multi-Robotic system is whether the system should be centralised or decentralised. It is further suggested in the same work that a single control agent dominates centralised models, where as decentralised architectures lack any such control rather the control itself is distributed amongst the agents.

This however must not be confused with the concept of homogeneity or heterogeneity of agents, which is a separate issue discussed in a later section of this document.

2.3 The Decentralised Approach

A decentralised model works in the absence of any central controller and is often characterized by having self-learning and self-organizing capabilities. Yao (1997) states that although a decentralised model has widely claimed benefits of Fault Tolerance, Natural Exploitation of Tolerance, reliability and scalability, yet there is no empirical or theoretical evidence of such a claim.

The decentralised approach emerged as a result of inspiration by social insects such as termite and ants. Melhuish *et al*. (1998, cited by Wessnitzer et al. 2001) demonstrated object sorting using minimalist Robots and algorithms. Stigmergy was used to achieve self-organising behaviour in mobile Robots. Change in behaviour of one Robot influenced the actions of other Robots. The model did not use any explicit communication mechanism and the Robots used information about/from their environment only to communicate and achieve puck sorting.

In practice, most systems do not conform to a strict Centralised or Decentralised architecture, rather behave in a hybrid way.

2.4 Cooperation Mechanisms

Beni et al. (1991) phrase the problem of interaction as follows: “the essence of the DRS (Distributed Robotic System) problem is to design a system that while composed of unintelligent units, is capable, as a group to perform tasks requiring intelligence; the so called ‘swarm intelligence’”.

Given some task by a designer, a Multiple Robotic System displays cooperative behaviour if, due to some underlying mechanism (the mechanism of cooperation) there is an increase in the total utility of the system. (Cao et al.1997)

2.5 EuSociality and Cooperation

McFarland (1994) classifies group behaviours into two types, Eusocial and Cooperative. Eusocial behaviour, as suggested, is exhibited by social insects (bees and ants) where individuals are not very capable but complex behaviours evolve out of their interactions.

In the same paper, McFarland says that cooperative behaviour is exhibited by certain intelligent animals; it is not motivated by innate behaviour but by an intentional desire to cooperate in order to achieve individual utility.

2.6 Formation and Marching

The concept of Formation has been borrowed explicitly from behaviours exhibited by Eusocial Insects and some higher-level animals. Trumpeter Swans are famous for migrating while forming a ‘\/’ shape (Figure 4). Wolves are best known for surrounding and hence trapping their prey. Ants frequently form straight trails while moving between their nest and food/prey, mainly owing to pheromone fumes. African soldier ants are best known for marching in formations of hundreds of thousands, and attacking prey that is 1000s of times larger than the ant itself.

Formation or Pattern Formation is defined by Bahceci et al. (2003) as the coordination of a group of Robots to get into and maintain a formation with a certain shape, such as a wedge or a chain.

Formation mechanisms occurring in nature have been studied intensively and have provided the grounds for their implementation in Artificial Intelligence. Solutions to formation can exist both in a Centralised as well as Decentralised fashion. Egerstedt *et al*. (2001) has demonstrated how to move a group of Robots in a desired formation over a given path. Koo and Shahruz (2001) have suggested a formation strategy for Un-manned Aerial Vehicles. They have adapted Centralised Architecture in which only leader has the decision-making capability while the rest simply follow. Robots are heterogeneous with leader equipped with cameras and other sensors absent in rest of UAVs.

A target assignment strategy for formation-building problem is described by Kowalczyk (2002), which assigns each Robot in a group of scattered Robots a target that has some meaning in the final formation. All the future movements of Robots are based upon these target locations.

A trajectory computation technique based upon kinetic energy shaping has been suggested by Belta and Kumar (2002, cited by Bahceci et al. 2003)

(a) (b)

Figure 4: (a) Trumpeter swans flying in 'V' formation, often seen during their migrations. (b) Ants formation around food.

2.7 Collision Avoidance

Mataric (1992a,b) looks at avoidance, aggregation and dispersion, combined to create an emergent flocking behaviour in groups of wheeled Robots. Albert uses a minimal speed/heading mechanism to demonstrate a collision avoidance strategy.

2.8 Summary

Starting with Beni's work in 1988, Swarm Intelligence and Swarm Robotics are areas that have widely attracted attention of scientists, most of them being those who were already involved in cooperative multi-robotics and artificial intelligence. The overview of present literature presents many problems; some already resolved, some partially solved and many of them yet be looked at by researchers.

A topic being out of scope of present work and hence not been discussed, is performance measures of a swarm, or rather the criterion for measuring the performance of a swarm. Lerman and Galstyan (2002) discuss the topic slightly, but it still needs attentions and is an area with tremendous research potential.

Chapter 3.
Simulating Swarm Behaviour

This chapter highlights high-level behaviours of Robots, which are governed by low level procedures discussed in Chapter 4.

3.1 Assumptions

For the project, certain careful assumptions have been made. However, practicality has been the major concern and any idea reckoned impractical for a real world hardware implementation has been avoided.

Robots are Homogeneous and each Robot is aware of its location within the world. On a macroscopic level, this might be implemented by GPS or some other suitable technology. On a microscopic level, such as for millimetre sized Robots, this mechanism may be implemented by using potential fields in case of decentralised approach, whereas in the case of a centralised architecture, an RTT (Real Time Tomography), CT Scan or any other suitable mechanism may be used. The controller can take over as soon as the first Robot finds the object.

3.2 Architecture

The algorithms designed during the project, and hence the overall architecture is hybrid i.e. it behaves both in a Centralised and Decentralised fashion. The model consists of Homogeneous Robots and starts in a Decentralised way. During the time that model behaves decentralised model, there is no Central Monitor or Controller.

The Robots follow their instincts, which have explicitly been defined in their governing algorithms. These instincts or behaviours, although a small set of simple rules, gives rise to an overall complex behaviour of the community.

3.3 Movement Models

These are low-level behaviours each Robot must show. These behaviours are not related or shared with other Robots, but may sometimes be affected by some surrounding individuals. These behaviours are responsible for depicting Ant like or Firefly like movement of Robots (or any other movement that may later be added); search for object, creating potential fields and following them.

3.3.1 Firefly like Motion

Firefly like motion exhibited here is more or less Brownian motion. At every point during its motion, a Robot can head in any direction, i.e. its next step can be at any angle from the set of 360 degrees, which makes Robots move along trajectories shown in Figure 5.

3.3.2 Ant like Motion

Ant like motion is complex than Firefly like; it is interesting, more realistic and can be customized according to requirements of a real Robot. The movement is governed by the factor `robotVisionSpan` which can manually be adjusted. Next step of a Robot following this type of motion is in a direction bound by region

$$\text{currentHeading} \quad + \quad \begin{cases} +\,\text{robotVisionSpan} \\ -\,\text{robotVisionSpan} \end{cases} \tag{1}$$

The decision for '`+ive`' or '`-ive`' direction is random, and is made by '`rt`' or '`lt`' NetLogo primitives, which will be discussed in detail in Chapter 4.

Ants tend to move along straight lines with random right and left turns, resulting in a movement shown in Figure 6.

Firefly like Motion

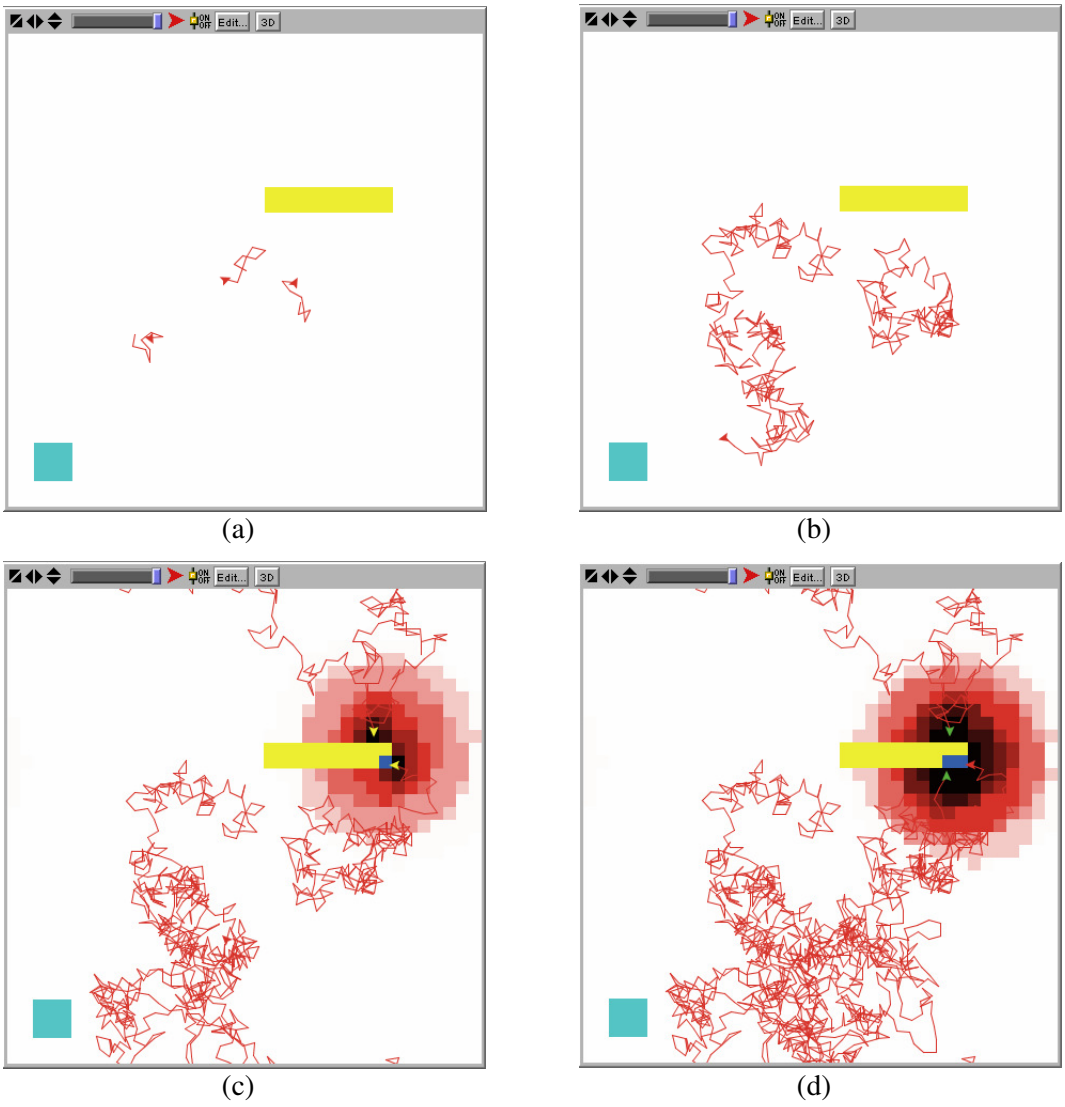


Figure 5: Firefly like motion is similar to Brownian motion. At every step during their motion, Robots can adapt a random heading and move ahead. The lines show large angles at which Robots turn during executing Firefly like motion. Above Simulation was run with only three Robots and shows that a wide area is covered by Robots while they search for an object.

Ant like Motion

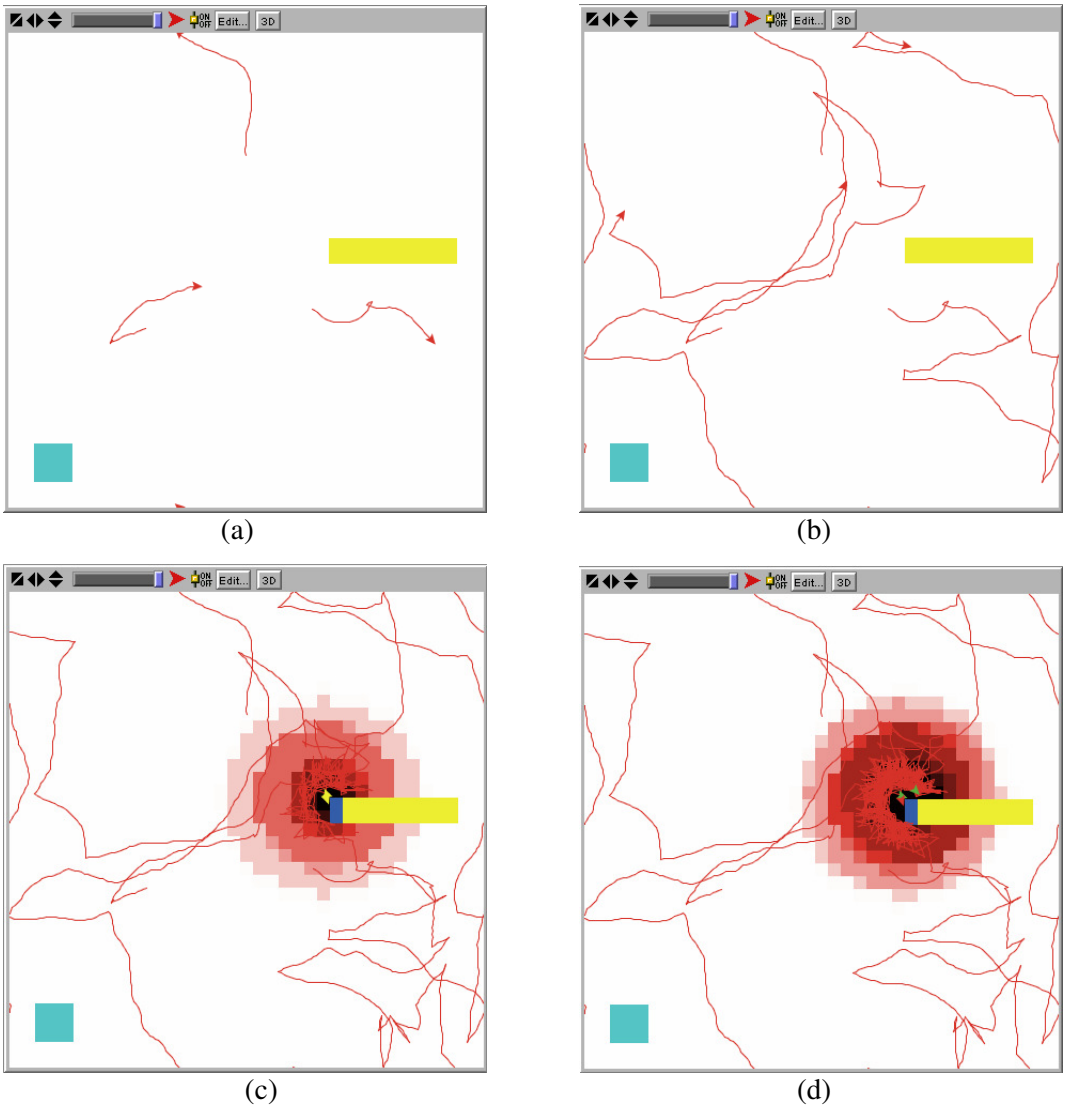


Figure 6: While moving, Ants have a limited field of vision. The angle of vision of Ants also limits the angle over which they may turn while moving. This results in long distances covered by ants before turning at reasonable angle to change the heading, resulting in a pattern with long lines and usually small turn angles. Note that a small factor of randomness has been introduced that makes Ants turn at sharp angles.

3.4 Cooperation

3.4.1 Swarm Behaviours

Robots show different behaviours, mainly Search Object, Try to Touch Object, Avoid Collisions with Other Robots, Surround Object and Transport it Back. Type of behaviour currently being exhibited can be found by Robot breed's 'own' variables corresponding to each behaviour, namely

- `foundObject`
- `foundField`
- `isMobile`

3.4.2 The Variable foundObject

Initialized to false, this variable shows whether the Robot has found any Object or not. When false, a Robot can be searching for object outside or inside the potential field which is represented by patch-variable '`field`', generated by another Robot. As soon as a Robot finds an object, this variable is set to true. Some important actions like generating a field depend upon this variable.

3.4.3 The Variable foundField

This variable is set to true as soon as a Robot enters potential field created by another Robot that has already found an object. A Robot with `foundField` true shows different behaviour from the rest of swarm. `fieldDefiance` may be used to change behaviours of these Robots. They may only move about while going downhill within the potential field (low value of `fieldDefiance`) or may have a randomness factor that allows them to distract from the aforesaid behaviour, thus increasing the chance to escape the field.

3.4.4 The Variable isMobile

All Robots are initially mobile (i.e. `isMobile true`). As soon as a Robot touches an object, it goes into immobile state (`isMobile false`). Only mobile Robots are allowed to move, with the exception being when the architecture turns into Centralised. When the power of Robots surrounding an object increases weight of the object, they start heading towards home, irrespective of value of the variable `isMobile`. This movement will be in a formation determined by the shape of the object.

Overall behaviour of the swarm is a result of multiple individual behaviours. Below follows a discussion.

3.5 Motional Behaviours

Individually, Robots show simple behaviours while they move around. These behaviours are:

- Before taking the next step, check whether there is a Robot, object or field present.
- If there is an object ahead, you do not need to move anymore. Just tell the controller that you have found an object. Try to lift the object.
- If there is a Robot ahead, change your heading and try to move again
- If a field can be sensed, enter it and try not to leave it. Go downhill.
- Try to touch object and avoid collisions with other Robots.

Motional behaviours appear as a result of low-level functions namely:

- `RandomHead`
- `HeadCarefully`
- `Turn`
- `FindField`
- `FindObject`
- `FollowField`
- `LookForObject`

3.6.1 RandomHead

Method: RandomHead

`RandomHead` and `HeadCarefully` methods together determine the basic Movement Model i.e. Firefly like or Ant like. N.B. any other models that may be added in future will be governed by these two methods.

`RandomHead` checks the variable `movementModel`. Possible values are

1; Firefly like Motion

2; Ant like Motion

For firefly like motion, a random heading is selected.

```
set heading random 360
```

This sets the variable to a random in the range

```
0<heading < 360.
```

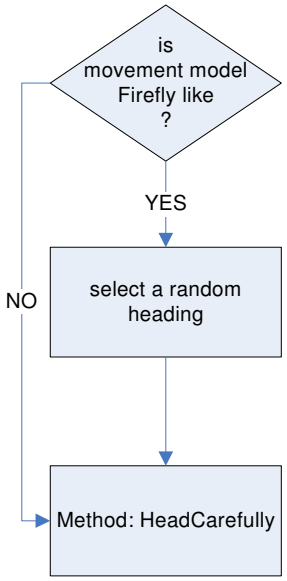


Figure 7: RandomHead plays vital role in *Robot*s' basic movement model. It is the core method that gives rise to collision avoidance Behaviour.

For Ant like motion, the method `HeadCarefully` is directly called without any change in heading. Heading for Ant like movement is taken care of in `HeadCarefully` itself.

3.6.2 HeadCarefully

This is the second method contributing to Robots' movement models. It starts by checking whether it is safe for the Robot to move (Figure 8). Instead of looking at safe conditions (Figure 9), let's define unsafe conditions in which a Robot cannot move.

- there is a Robot ahead
- there is an object ahead, an area of which is not being looked at.
- there is an object ahead, an area of which being looked at.

These conditions are checked with quite ease in a single line NetLogo statement (Figure 9).

An object is identified by yellow coloured patches, while an area of object being looked at by a Robot turns blue.

If it is not safe for a Robot to move ahead, it selects a random heading for its next step forward (but does not move, until the condition for safety is checked again) and returns. This causes Robots to turn at large angles while observing Ant like movements in which Robots usually turn at small angles.

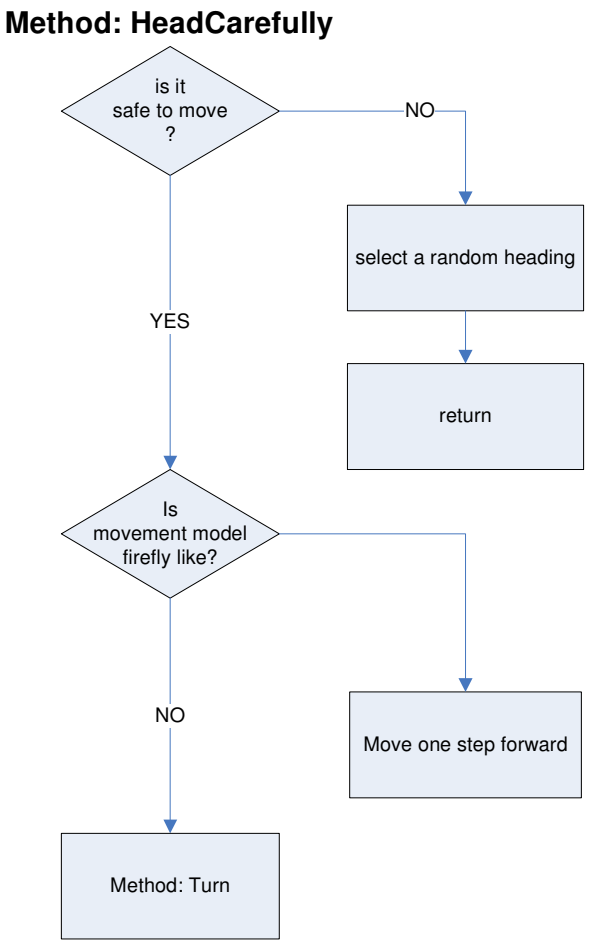


Figure 8: HeadCarefully forms the base of collision avoidance. The function also contributes to deadlock prevention under normal conditions. The system does not prevent deadlocks, but allows them to occur and the algorithm takes care of them

```
ifelse(any? turtles-on (patch-at-heading-and-distance heading 1))
or ( pcolor-of patch-at-heading-and-distance heading 1 = yellow)
or ( pcolor-of patch-at-heading-and-distance heading 1 = blue)
```

Figure 9: NetLogo statement for collision avoidance; ifelse requires two blocks of statements, one for the case when condition is true, and second when the condition is false. Note the ease in which a condition for very complex behaviour is checked.

3.6.3 Turn

Only valid for Ant like movement model. The method uses random number generator and based on the result (random 100 mod 2), takes decision to head right or left using:

```
rt (or lt) random
RobotVisionSpan
```

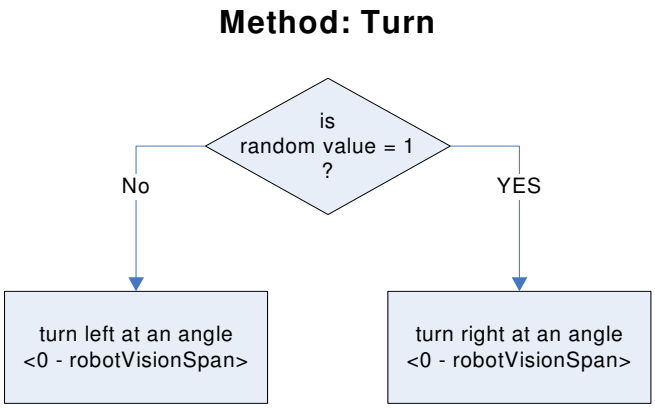


Figure 10: Turn; The method sets heading within a span of current `heading +` *Robot*`VisionSpan`

Turn limits the turning angle of ants directly and vision span indirectly and can be controlled externally by the variable `robotVisionSpan`.

Please note that `robotVisionSpan` has no effect in case of Firefly like motion.

3.6.4 FindField

The method `FindField` looks at the patch immediately ahead for any trace of potential field. `field` is a patch variable, i.e. potential field exists in the world as a variable whose value is set by Robots (Figure 11). If a Robot finds `field`, it simply tries to move downhill by calling `FollowField`. Again, the check for field is a single NetLogo statement:

```
if field-of patch-ahead 1 > 0
```

The `foundField` variable of a Robot that has entered or touched the field is true. `foundField` apparently is not significant, but the whole idea of the term 'Incremental' perception depends upon this variable.

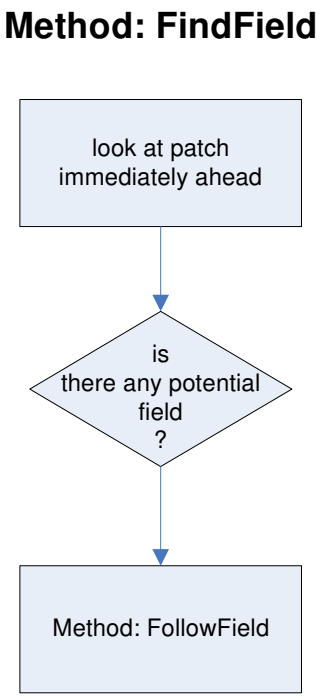


Figure 11: FindField, simple yet critical.

Any Robot that finds an object with this variable already set to true, will believe that it is looking at a segment of the object, a part of which is already being looked at by another Robot. The Robot informs controller about its location and thus increments the overall perception.

It was mentioned earlier that all the variables used in present model could be mapped to a real life parameter and all behaviours to real life behaviours. `FollowField` can be implemented in hardware as a means to detect potential field, which may simply be a radial Electromagnetic Field.

3.6.5 FollowField

This method sets the heading of calling Robot in a direction of increasing potential field. It calls the method `HeadCarefully` (already discussed) so that the Robot may continue to move according to its movement model.

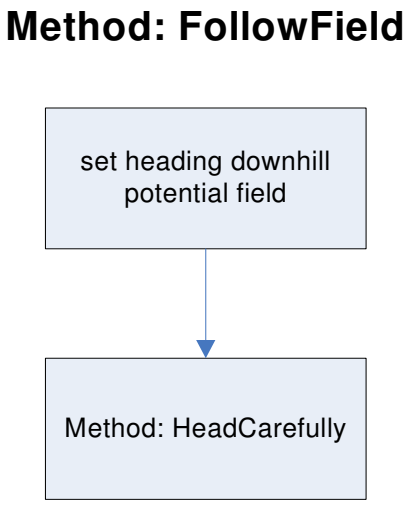


Figure 12: FollowField relies upon accuracy of low level routine HeadCarefully

It will later be seen that potential fields exist merely as a variable, a patch variable to be precise. Distribution of this variable as a function of distance from a calling Robot results in an attractive force which attracts every near by Robot towards the Robot that created the field. It may or may not yet be known whether the present object is the required object or not.

3.6.6 LookForObject

NetLogo world is divided into patches, which have been given a white colour. An object is characterized by a yellow coloured patch. Thus `LookForObject` is actually looking for a yellow coloured patch, which obviously will be an object.

Once an object has been found, some important changes are needed; the Robot needs to become immobile, generate a stationary field around it, be aware that it has found object, try to lift the object to bring it back, tell the controller that it has just found an object, thus turning the whole model into a Decentralised Architecture.

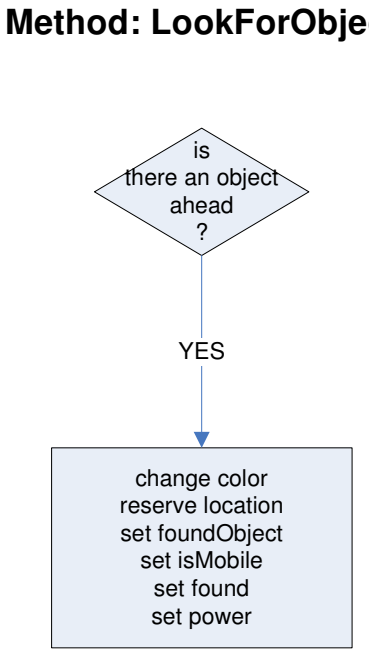


Figure 13: `LookForObject` affects the largest number of variables. Once a Robot finds an object, the behaviour of *swarm* becomes centralised.

3.6.7 FindObject

Find object behaves differently for Firefly like and Ant like movement. A firefly looks all around it i.e. it eyes all 360 degrees around itself, while an ant can look only in a small field of vision (Figure 14).

A firefly looks around it to see if an object is present (by calling `LookForObject`) and continues to look around until either the object is found or the loop (`repeat for 0, 90, 180, 270 degrees heading`) breaks, which ever is earlier.

For the Ant like movement, method `Turn` is called which sets the direction in which Robot is looking in, and then calls `LookForObject` to check whether there is any object present in that direction.

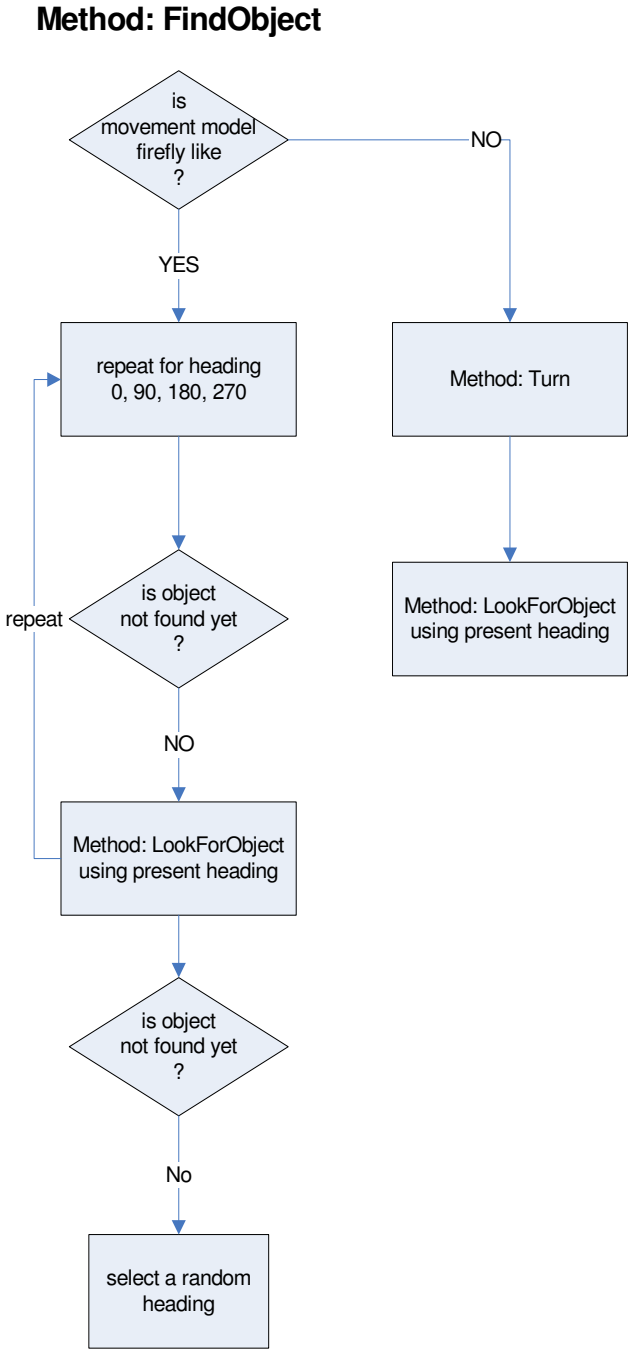
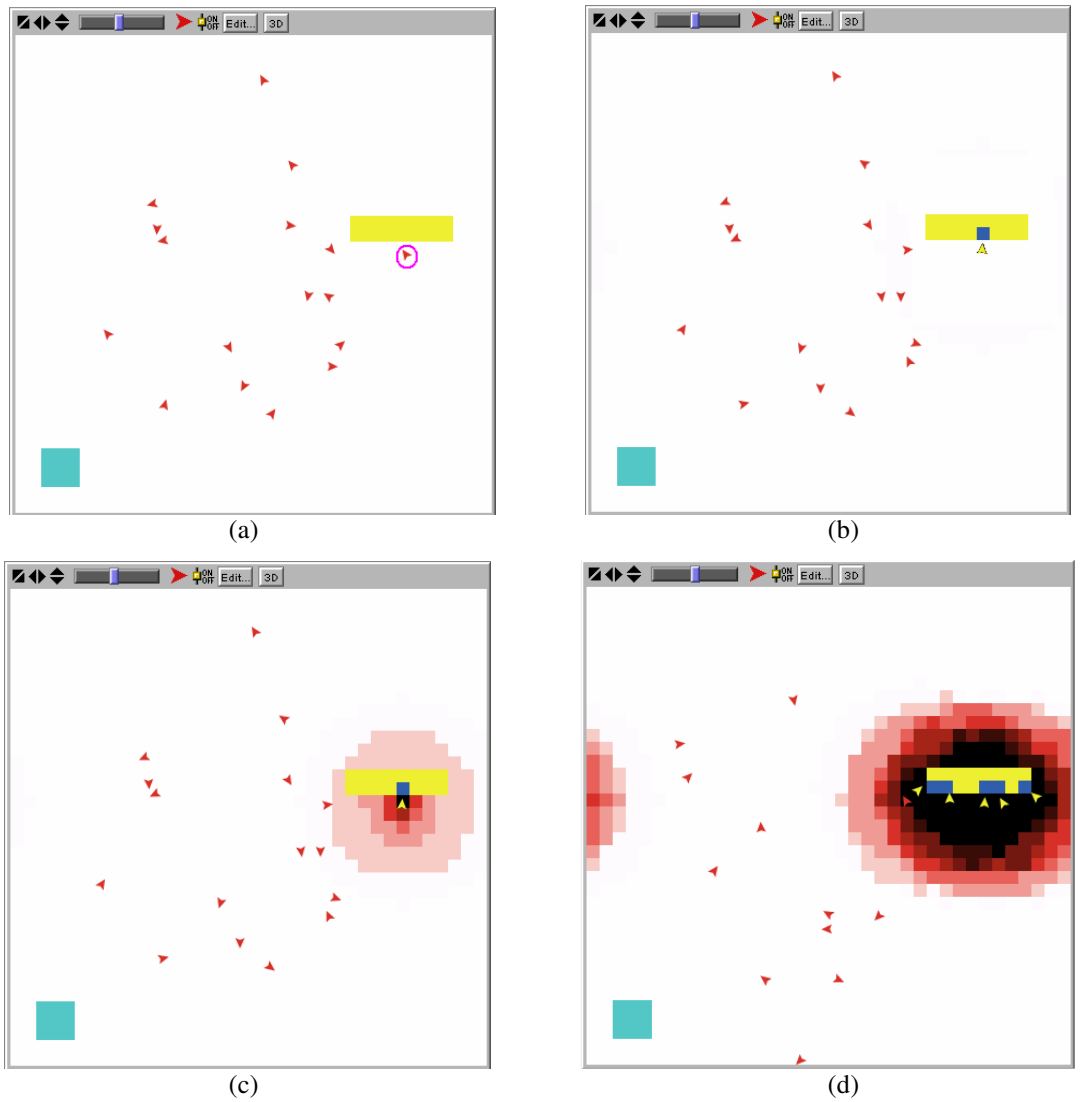


Figure 14: `FindObject` gives rise to low-level complex behaviours. Firefly like movement mainly depends upon the algorithm used in this method. If used carelessly, the code for firefly like movement can convert the movement into a Holonomic movement.

3.7 Object Detection

The world is a grid of 18 x 18 square patches that are white in colour. A yellow coloured patch characterizes an object, so the Robots are actually looking for yellow patches in the world. As soon as a Robot finds an object, it stops and continues to look at the object. The area of object that a Robot is looking at turns blue, and will not be recognized as an object by another Robot. Robots that have found an object also create a potential field around them, thus telling other Robots about presence of an object. Robots try to encircle an object while staying away from other Robots.



(a) (b) (c) (d)

Figure 15: a. A Robot approaches object. b. It turns the patch ahead blue so that other Robots may not stick to this patch. c. Robot generates a potential field around it. d. Other Robots enter field and eventually find the object.

3.8 Collision Avoidance

While moving around, Robots try not to collide with other Robots. Before taking a step forward, a Robot first checks whether it is safe to do so by looking on the patch that it intends to step on to. If there is another Robot present, it simply changes its direction and attempts to move again. The technique is similar to CSMA/CDs collision detection but apparently has a risk of deadlock, which might arise in an area that is over populated by Robots. As such, not even a single deadlock has been observer over more that 1000 simulations.

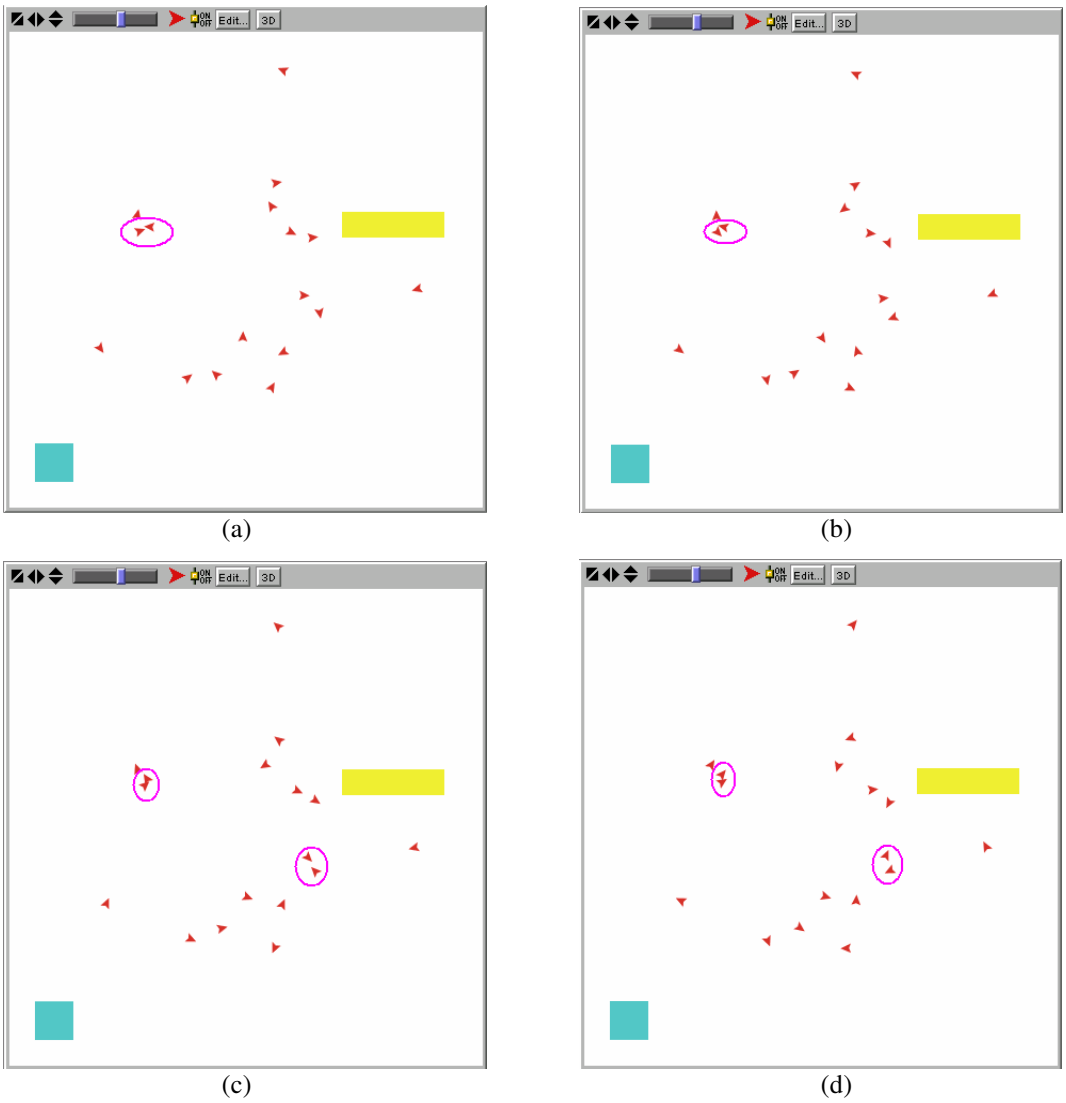
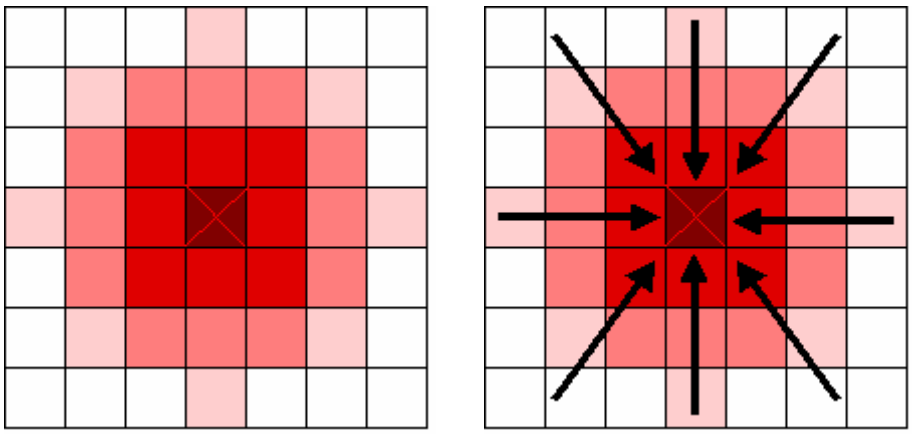


Figure 16: a. Robots move ahead while creating a possibility of collision if they move any further. b. Heading is changed and they move forward. c. Possibility of collision again, in the same pair and in another pair of Robots. d. Both pairs avoid collision. The second pair moves in two completely different directions.

3.9 Potential-Field Methods

The present architecture uses potential field methods to inform other Robots about a prospective target object. Potential fields have extensively been used in Robotics, details about the approach for obstacle avoidance etc can be found in Koren et el. (1991).

Patches in the world have a variable `field` that corresponds to a potential field in real life. Field is normally '0', unless a Robot finds an object. It then generates a field cantered at its present location. Any Robot entering the field will follow it, knowing that there is another Robot around which is looking at the target object.



(a) (b)

Figure 17: a. Patch marked X is the origin of field. Stronger the field, darker is the colour. b. At every point in the field, a Robot experiences an attractive force, always directed towards center of the field. Robots may overcome this attractive force owing to the randomness introduced by fieldDefiance.

3.9.1 Field Strength and Radius Factor

The potential field generated by a Robot has two attributes, *strength* and *radius*. The potential field here is actually an inverse potential field i.e. Robots move in a direction of increasing field. Strength of potential field is determined by

```
(field +  round (10 / (1 + distancexy tempx tempy)) )
```
(2)

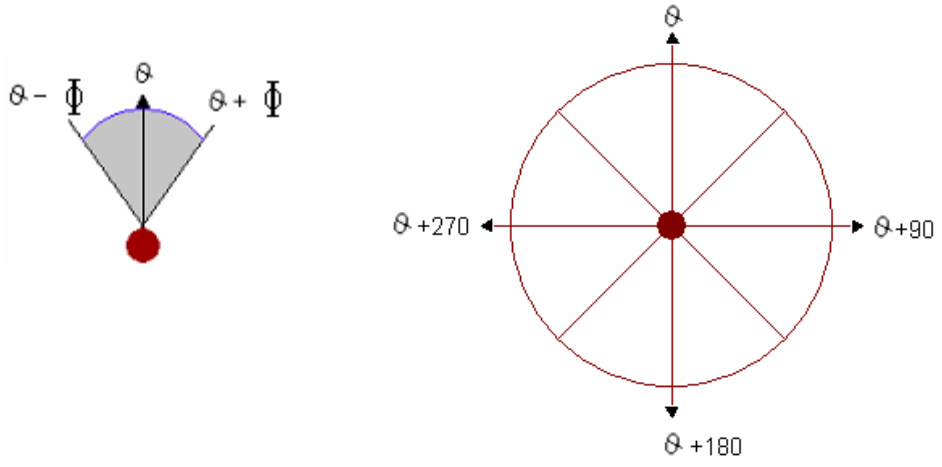
This gives a field, which is a factor of distance from the originating point.

3.9.2 Field Defiance

Once inside the field, if Robots only follow field, they would end up near the first Robot that generated field, and stop there because that would be strongest point of field. In order to allow Robots to have an element of randomness within the field, `fieldDefiance` factor has been introduced. The variable `fieldDefiance` defines the scale by which the Robots would disobey the rule of following the field.

3.10 Robot Vision Span Factor

The variable `robotVisionSpan` differentiates Ant like motion from Firefly like motion. Vision span for Ant like motion defines the total angle of vision of a Robot which is $\theta \pm$ `robotVisionSpan`. This is also the angle over which an Ant like Robot can turn. Only exception to the rule is when a Robot feels a possibility of collision with another Robot. Only in this particular case a Robot may turn at any angle from 0° to 360°. Heading angle is chosen at random from the range of possible angles, and a Robot may turn in positive or negative direction at that angle.



(a) (b)

Figure 18: a. Ant like Robots have limited angle of vision and turn. θ is the present heading of Robot, while Φ is externally defined `robotVisionSpan`. Larger the value of this variable, sharper turns can the Ants take. b. A Robot with Firefly like motion can look around and turn at any angle it likes. Thus the turning angle and angle of vision is the whole set of 360˚. This results in a Brownian motion for firefly like Robots.

The Variable `robotVisionSpan` does not affect firefly like motion. Reason being that firefly like motion is appears only when a Robot is given freedom to move in any direction it feels like, and thus Firefly like Robots by default have a `robotVisionSpan` of 360°. Please note that this span for Firefly like Robots is not explicitly defined but emerges as a result of `RandomHead` Method.

3.11 Object Weight and Robot Power

An object has some weight that can be configured externally. Along with this, each Robot has some power. When power of all Robots equals or exceeds an objects weight, only then will the *swarm* be able to bring the object back home.

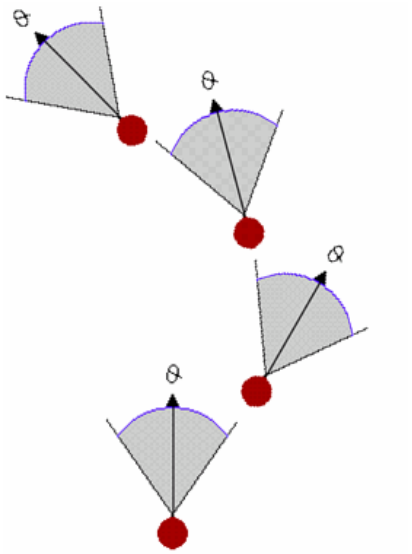


Figure 19: Ant like movement of Robots when they turn at maximum angles i.e. θ + Φ.

3.12 The Centralisation Factor

As mentioned earlier, the present model has Hybrid architecture. It conforms to a decentralised model until the first Robot finds an object, which is when the controller wakes up. Any Robot that touches an object immediately informs the controller about its location. The controller keeps collecting the information and extracts the shape of object.

3.13 Selection of The Leader

Robots in the model are homogenous, i.e. there is no structural difference between them (same set of variables). Hence the selection of leader is arbitrary, depending upon who finds the object first. The first Robot to find the object becomes the Leader, and will determine the direction of home when the *swarm* has enough power to transport the object back.

3.14 Bringing Object Back Home

As soon as the power of Robots increases the object's weight, Robots lift it and try to bring it back home. A potential field that exists as a patch-variable `wayHome` governs their journey.

3.15 Summary

This chapter describes in detail the low-level NetLogo procedures that give rise to high-level behaviours showed by Robots. NetLogo procedures depend upon some variables that act as flags, and control entry into their respective procedures. NetLogo allows manipulation of these variables by agents that do not possess them, i.e. Robots can change patch variables.

Chapter 4.
NetLogo and Implementation

Written in Java, NetLogo is the one of the series of next generation multi-agent modelling languages that started with StarLogo, and is a programmable modelling environment for simulating natural and social phenomenon. It is particularly well suited for modelling complex systems developing over time. Designers can give instructions to thousands of independent agents all operating concurrently. (Wilwnsky 1999)

This chapter defines the low level procedures that give rise to High-Level behaviours discussed in chapter 3. It discusses the NetLogo primitives used and looks at the routines that have been developed for the sake of this project.

4.1 Structure

The model is based upon its participants, namely Robots, Patches and the Object, and the set of rules for each of these participants. These rules govern individual behaviour of the participants, as well as specify the way in which these participants will interact with each other. The set of rules, defined as NetLogo functions, is discussed in detail in a later section of this chapter, while the next section looks at the participants.

4.2 Participants

There are two main participants as for as the code for model is concerned, namely Robots and Patches. Loosely speaking, the 'world' can also be classified as another participant. Technically, Robots are NetLogo turtles defined as a Robot breed, while patches the NetLogo patches. Both have a set of variables associated with them that enables the model to show different behaviours. The World is a set of global variables that facilitates interoperability of Robots and Patches. All these participants exist in the Global Space.

4.2.1 The Robot

Robots are declared as a Breed, which is a mobile agent. NetLogo defines agents as beings that carry out some instructions and work in parallel to other agents. Breeds are groups of mobile agents that have same characteristics and follow the same set of rules. While the use of breeds refines code on one hand, its major benefits are the vast set of primitives that are associated with it and versatility in which multiple breeds can be handled within the same model. Since the present model has hybrid centralised-decentralised architecture, breeds have been used keeping in mind some extensions to models discussed in Chapter 6.

Robot

`foundObject`	Initialized to false, when a Robot touches an object, `foundObject` is set to *true*.
`foundField`	Robots entering potential field generated by other Robots turn this variable *true*.
`isMobile`	A Robot is mobile (`isMobile` *true*) until it finds an object.
`isLeader`	When power of Robots surrounding an object becomes greater than object's weight, first Robot to find the object becomes leader (isLeader *true*).

Table 1: A Robot actually is an entity with the above-mentioned attributes. Each of these variables is scalable to a real life property and can easily be mapped while testing the algorithms on a real Robot.

Each Robot is aware of its location in global space, which is available as two built in turtle variables *xcor* and *ycor*. Other built in variables include shape, size, colour etc and can easily be modified to show traits visually. A rich set of primitives is also available, some of which (Table 2) have frequently been used.

`turtle` primitives
set heading; downhill; forward; patch-at-heading-and-distance; rt

Table 2: Some of the most frequently used turtle primitives. NetLogo primitives are usually self-explanatory e.g. *`ask patch-at-heading-and-distance <heading> <distance>[]`* asks a patch at given heading and distance to execute a given piece of code.

4.2.2 Patches

The NetLogo world is divided into patches, which are a special type of agent, as alive as turtles but immobile. Just like any other agent, a patch can also execute a set of code and show certain behaviours. This is one of the features that distinguish NetLogo from other modelling languages. The concept is very much realistic allows to define laws for the Global Space. Each patch has its coordinates and 0,0 is the origin. Below are variables that are used for patches in the present model.

Patch

`field`	This is the Potential field that Robots generate when they find an object. A Robot in a potential field experiences an attractive force towards center of the field.
`oldField`	Potential field surrounds a Robot and moves with it. When a Robot that has generated field around it moves, the patches behind are set to the previous value of field.
`wayHome`	Way home comes into existence when the *swarm* decides to move the object back home.

Table 3: Important concepts such as potential fields, which are difficult to model in languages like C or Matlab, can easily be integrated in a NetLogo model. Use of only one variable along with certain NetLogo primitives can result in complex behaviours and simulation of physical laws relating to attractive and repulsive forces.

4.2.3 Global Variables; Globals

These are global variables accessible to Patches and Robots (and to any other agents added to the model in future) and facilitate their interoperability. Although they do not contribute to any behaviours directly, but are vital as the whole idea behind model is cooperation and interoperation of agents, which are heavily dependent upon these global variables.

Global Variables

`found`	A Robot that finds an object also sets the global found true to enable signal generation.
`locx`, `locy`	Robot's coordinates when it finds an object.
`objx`, `objy`	Location of object. Used to redraw object.
`canTransport`	True when power of Robots equals or increases object's weight, false otherwise. When true, model turns into a centralised architecture.
`lHead`	Variable used in centralised model to set the heading of whole *swarm* towards home.
`power`	Every Robot that finds object adds to the power of Robots surrounding object.
`timeCheck`	Variable that allows entry to routine that writes convergence time to a file.

Table 4: Important concepts such as potential fields, which are difficult to model in languages like C or Matlab, can easily be integrated in a NetLogo model. Use of only one variable along with certain NetLogo primitives can result in complex behaviours and simulation of physical laws relating to attractive and repulsive forces.

Some of these variables are used as check points and flags for entry into a code segment. For example the variable `found` controls entry to the method `Signal`, a function that generates Potential Field around a Robot. Others contribute to turn architecture into centralised, for example `lHead, power.`

Chapter 6 suggests how the number of these variables can be reduced to give a minimal overhead model that is equally efficient.

4.3 Movement Models

Robots show two movement models, Firefly like and Ant like. These movements emerge from low level behaviours discussed below. They are governed by certain parameters (e.g. RobotVisionSpan) and are heavily affected by a change in value of these parameters.

4.3.1 Ant Like Movement; Procedure Turn

While behaving as ants, the Robots have a finite field of vision with a limited angle of vision and turning angle, which can be controlled externally through the variable `robotVisionSpan`. The method Turn is the key to this type of movement.

Turn

```
let fac ( random 100 mod 2 )
ifelse fac = 0
[ rt random RobotVisionSpan ]
[ lt random RobotVisionSpan ]
```

Table 5: `rt` and `lt` primitives are used for right or left turn. `random` uses Java's strict math Library.

The method decides left or right turn (in degrees) by using random number generator. `rt` and `lt` (turn right, left) primitives ease the code for Ants movement.

4.3.2 Firefly Movement; Procedure RandomHead

Fireflies, as mentioned earlier, show a Brownian motion. At every step during their motion, they may turn at any angle and head forward. Their angle of vision angle is also 360°, so they may look all around them for an object or field.

```
RandomHead
```

```
ifelse movementModel = 1
[
  set heading random 360
  HeadCarefully
]
. . . . . .
```

Table 6: `movementModel` = 1 means firefly like and 2 means Ant like motion. `HeadCarefully` is discussed in detail in section 4.4.7.

4.4 Procedures

NetLogo comes with a rich set of instructions called commands. These commands can either be used individually to ask agents to behave in certain ways, or can be grouped together to give complex behaviours.

Each NetLogo procedure starts with a 'to-' and ends with an 'end' statement.

4.4.1 Setup

Before a model can be started, setup creates the environment for it. It is conventional to call the first executed function setup, although any name can be used.

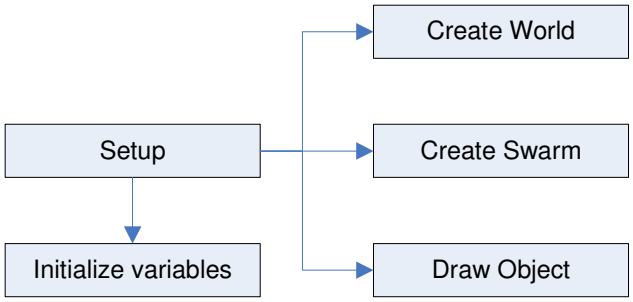


Figure 20: Procedure Setup initializes some variables itself, and calls initialization routines for the rest.

Setup initializes certain variables and then calls three other functions, namely `CreateWorld`, `CreateSwarm` and `DrawObject`.

`CreateWorld` and `CreateSwarm` further initialize specialized variables private for patches and Robots. `DrawObject` creates the object according to the pattern already specified by user externally.

Variables affected by Setup and functions called

Setup	CreateWorld	Create*Swarm*
found	pcolour	colour; size
canTransport	objx	isMobile
reset-timer	objy	foundObject
timeChek		foundField
dra		isLeader

Table 7: Setup is the method that affects largest number of variables, directly in its code, and indirectly by calling other methods. Variables initialized by setup itself are global variables.

4.4.2 Go

Go is a forever procedure, i.e. one that runs over and over again until a condition to stop its execution occurs. It forms the major body of model and is a function that checks certain conditions to call methods. These conditions and methods govern the overall behaviour of participants. Go uses low-level code segments to give certain high-level behaviours such as Object Search, following a field, attracting other Robots, bringing object back.

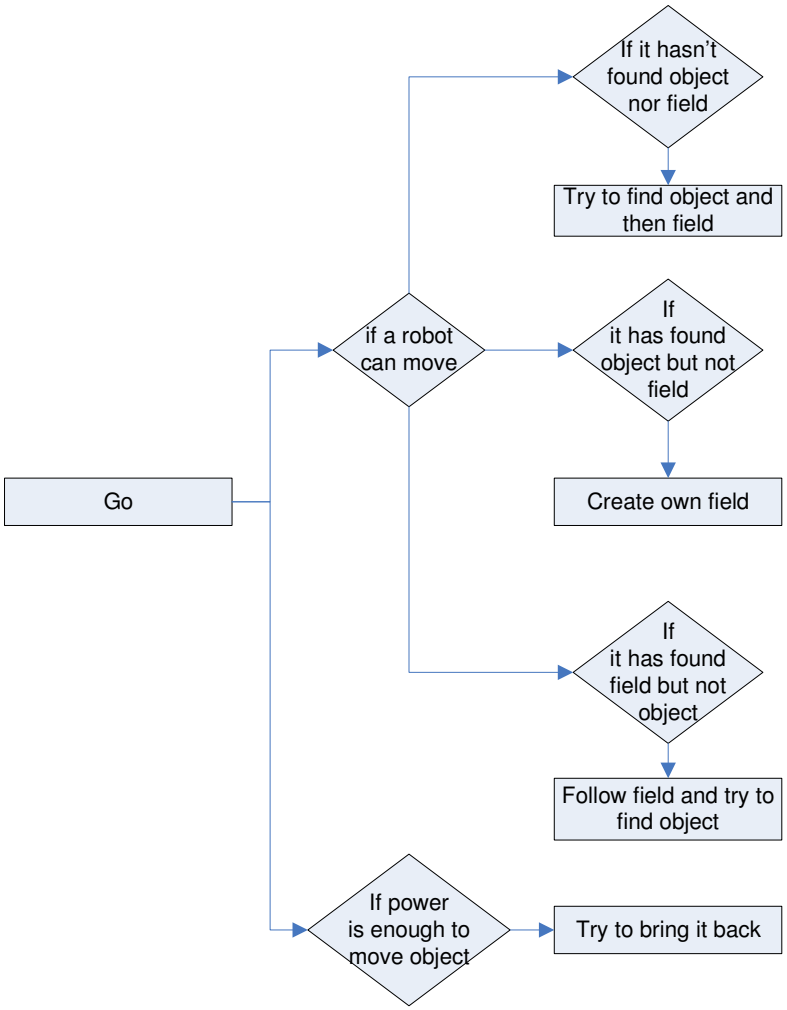


Figure 21: Go implements the main algorithm and his heart of the model. Basic Algorithm can be seen in the diagram above, while details can be seen in Appendix A.

Go implements the basic algorithm and facilitates the high-level behaviours by checking patch and Robot variables and taking decisions based on the values of these variables.

4.4.3 Object Search

Object search is carried out by two methods, a low level routine `LookForObject` and a high level routine `FindObject`. `LookForObject` is called both by fireflies and ants, while `FindObject` behaves differently for both movements.

4.4.4 LookForObject

Look for object looks at the patch ahead at unit distance an in the present direction. Heading of Robot must be adjusted before calling this method. Thus `FindObject` (or any other method that decides to use it) uses set heading before calling this method.

`LookForObject`

```
if pcolour-of patch-at-heading-and-distance heading 1 = yellow
```

Table 8a: patch-at-heading-and-distance checks the patch at a given distance and in a direction already adjusted, for any subsequent statements.

If condition of if stands true, Robot and global parameters are altered to show that the present Robot has found an object.

`LookForObject`

```
set pcolour-of patch-ahead 1 blue
set foundObject true
set isMobile false
SetLocaion
set found true
set power power + rPower
write the current time to file: "indstats.txt"
```

Table 8b: Global and Robot parameters adjusted by `LookForObject`. To appreciate the ease of File I/O please refer to Appendix A.

4.4.5 FindObject

A high level routine that first checks what type of movement the model is following, and then calls another method accordingly. In both cases, it relies on `LookForObject` to actually check for object.

`FindObject`

```
if firefly motion
    foreach [0 90 180 270]
    [
   if not foundObject
       [ set heading ?   LookForObject  ]
    ]
otherwise [   Turn   LookForObject  ]
```

Table 9: For firefly motion, heading is set explicitly by *foreach* list. For Ant like motion, Turn adjusts the heading.

4.4.6 RandomHead

`RandomHead` calls `HeadCarefully` after adjusting a random heading in case of firefly like motion. For ants, it only calls `HeadCarefully`. Repeat can be used to make ants cover a certain distance before turning.

4.4.7 Collision Avoidance: HeadCarefully

Any Robots that come close to each other experience a repulsive force, direction of which is random, if after moving one step forward, there is a possibility of having more than one Robots on one single patch. `HeadCarefully` checks the condition for collision.

`HeadCarefully`

```
ifelse
(any? turtles-on ( patch-at-heading-and-distance heading 1) )
or (pcolour-of patch-at-heading-and-distance heading 1 = yellow)
or (pcolour-of patch-at-heading-and-distance heading 1 = blue)
```

Table 10: A yellow coloured patch is an object or a segment of an object that is not being looked at by any Robot. As soon as a Robot starts looking at a patch, it turns blue in colour.

If it is not safe to move, Robot selects a random heading only but does not move. Random heading is selected so that it may attempt to move in one of the subsequent iterations.

If it is safe to move, a Robot moves one step forward (fd 1) in case of firefly like motion, and 0.25 step (fd 0.25) in case of Ant like motion.

4.4.8 FindField

`FindField` is similar to find object. The difference being that find object looks for a yellow coloured patch while `FindField` scans the patch ahead for field.

`FindField`

```
if field-of patch-ahead 1 > 0
```

Table 11: Another powerful NetLogo primitive '-of'. Usage:<variable name>-of [agentset]. The above statement checks value of field for the patch immediately ahead in a given direction. heading of Robot must be adjusted before calling the method.

4.4.9 FollowField

The model uses inverse potential field method i.e. Robots move uphill instead of going downhill in a field.

`FollowField`

```
  set heading uphill field
  HeadCarefully
```

Table 12: uphill [variable] selects a patch from surrounding patches such that the value of variable is greater than the present value. Heading of Robot is then adjusted in the direction of that patch.

4.4.10 Signal

Signal generates a potential field for a stationary Robot that has found an object according to the following equation:

$$\Phi_n = \Phi_c + R\left(\frac{10}{1+\Delta_{pn}}\right) \quad (3)$$

Φ_n is the new value of `field` for n^{th} patch, Φ_c is the current value of `field` of present patch, Δ_{pn} is the Euclidean distance from this patch to n^{th} patch, R is rounding function that rounds input value to nearest decimal.

Signal[x y]

```
ask patch-at x y
ask patches in-radius sigRadius [
set field(field +  round (10 / (1 + distancexy tempx tempy)))
ifelse((pcolour = blue) or (pcolour = yellow))
[] [set pcolour scale-colour red (10 * field) 90 10]
]
```

Table 13: A stationary Robot calls `Signal [x-coordinate y-coordinate]`. `in-radius<>` selects all patches in a given radius and executes any subsequent instructions.

Chapter 5.
Statistical Analysis and Results

A set of experiments was designed and the model was run and tested several times for different system parameters. Two factors are significant while analyzing the model. Firstly, as major objective is perception of the object and shape extraction, effect of number of Robots on relevance of shape extracted to original shape of object was studied. Secondly, statistical data to find out effect of different factors on convergence time was collected and analyzed. It is worth noting that the major concern i.e. shape extraction was easy to be analyzed as it only depends upon two factors, the Swarm population and spatial distribution. Convergence time, as can be see in the following sections of this chapter, depends upon every system parameter.

5.1 Scalability

Scalability implies that a system or algorithm should give optimal performance, regardless of the swarm population. A scalable system would offer the same performance even if an arbitrary number of agents are added or removed from the system. A careful definition would subject scalability to addition of an arbitrary number of agents, while robustness to removal of an arbitrary number.

McLurkin (2004) suggests that scalability also requires that algorithms do not scale in running time or in memory space as a function of n, the total number of Robots. The

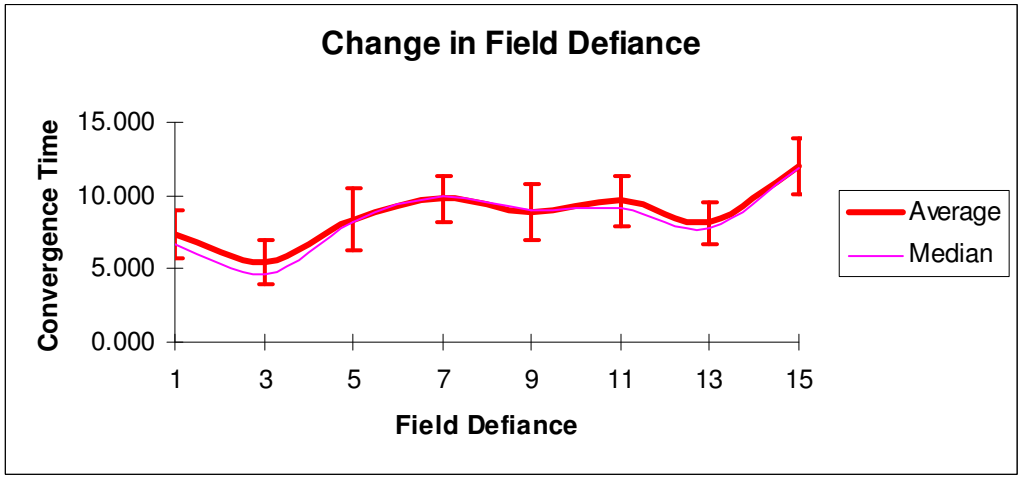
graphs below show that performance of the system actually improves with the increase in number of Robots.

Convergence time has been averaged over values taken from 50 simulations.

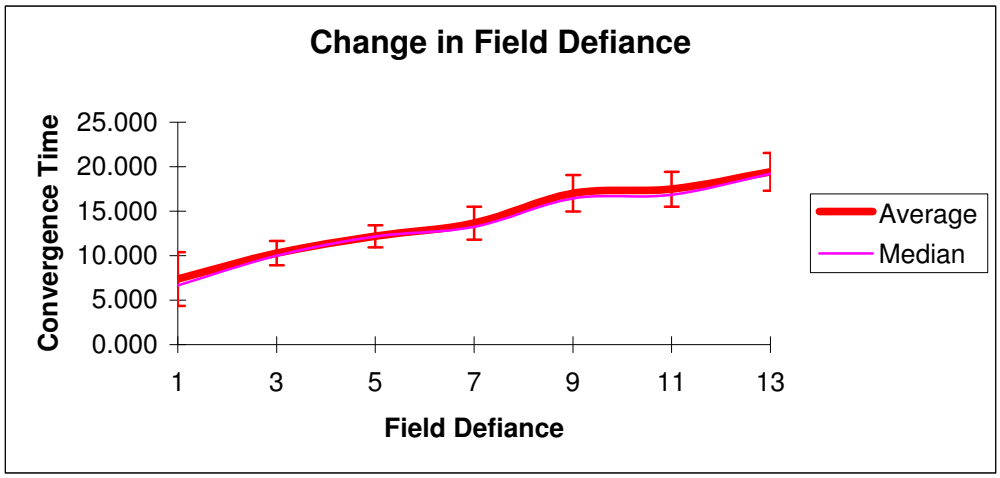
5.2 Effect of Different Factors on Convergence Time

All the statistics below are collected from data averaged over at least 150 simulations for each experiment.

5.2.1 FieldDefiance



(a)

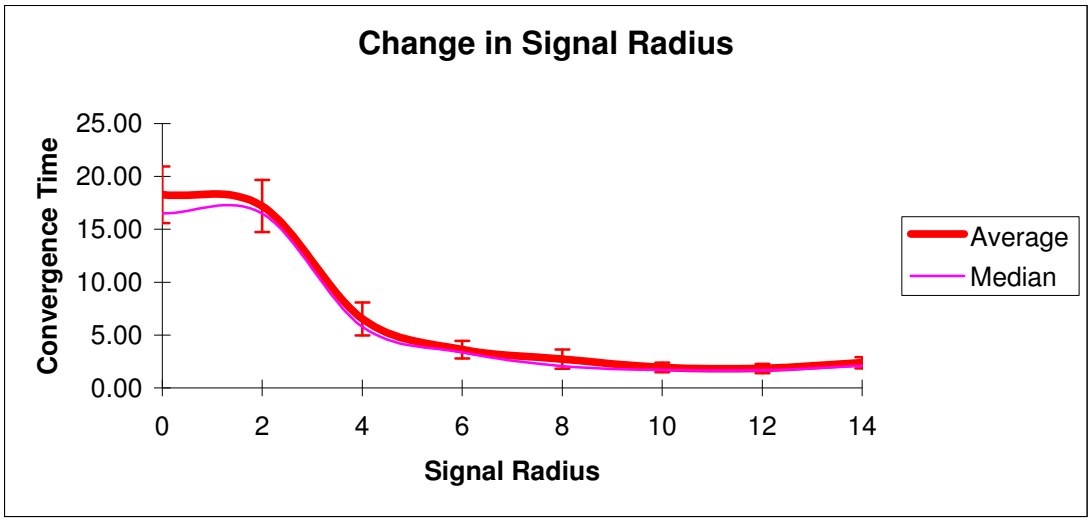


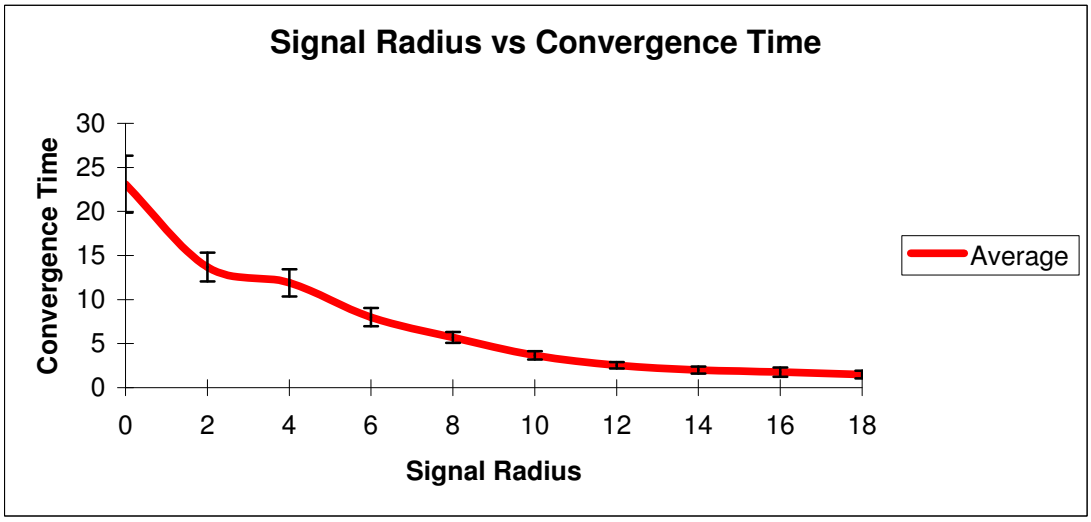
(b)

Graph 1: a. Field Defiance vs. Convergence Time in Firefly like Motion. b. in Ant like Motion

Graph 1a shows that there is an irregular increase in convergence time as field defiance increases for Firefly like Motion, while a steady increase is seen in the case of Ant like Motion. The range of Convergence Time varies from 4.5ms to 10ms in case of Firefly Motion, while for Ant Motion it is distributed in a range of 7.5-20ms.

5.2.2 Signal Radius

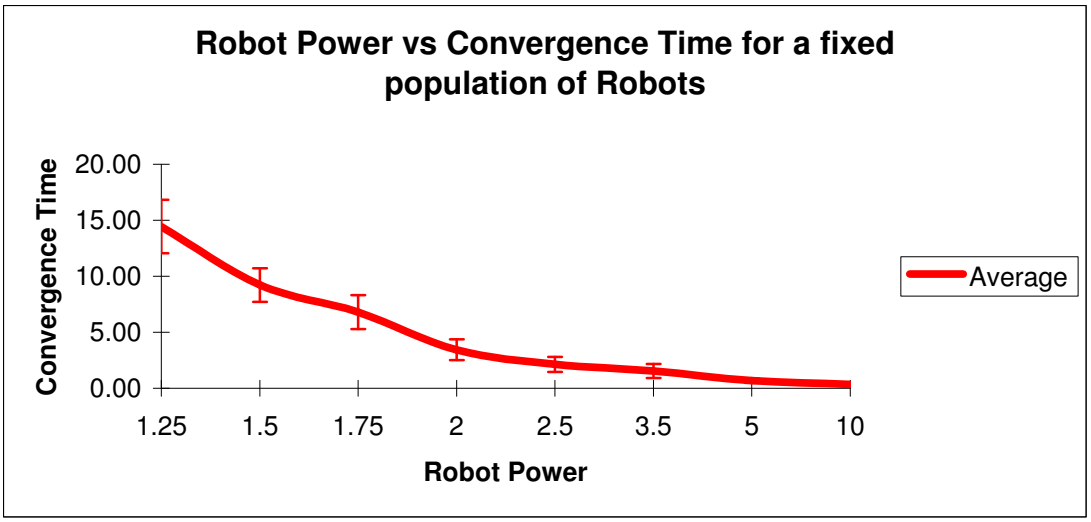




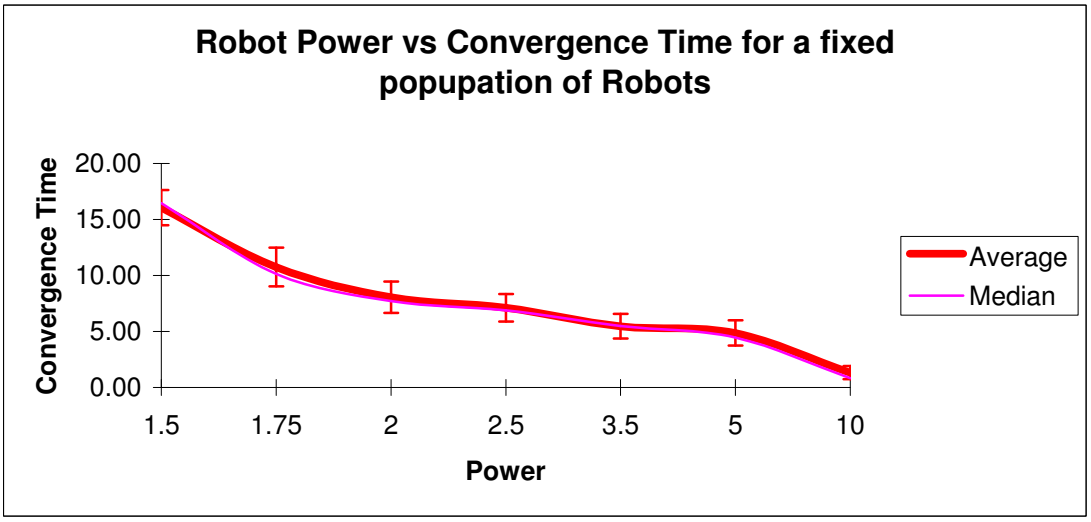
Graph 2: a. Signal Radius vs. Convergence Time in Firefly like Motion. b. in Ant like Motion

Signal Radius, as expected, reduces convergence time.

5.2.3 Robot Power



(a)



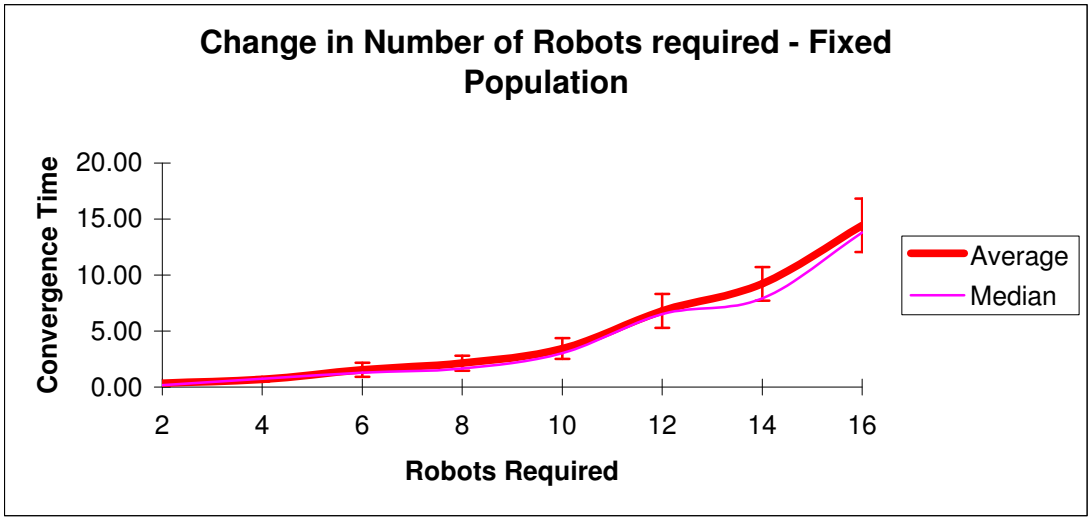
(b)

Graph 3: Robot Power vs. Convergence Time in a. Firefly like motion. b. Ant like motion

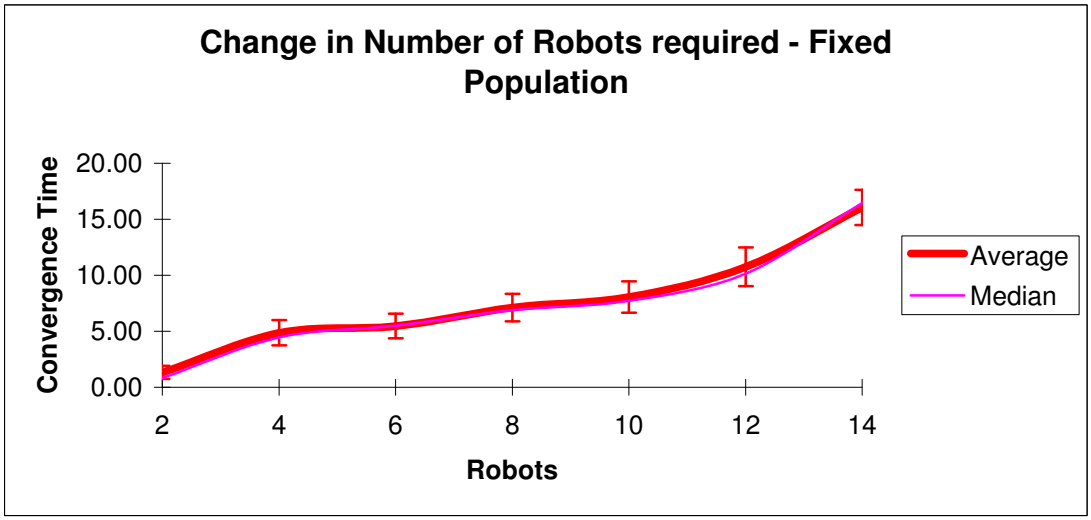
Robot Power '1.25' has been ignored in case of Ant like motion because it took unreasonably long time for the swarm to converge.

The two graphs are identical, with Firefly like Robots converging earlier than Ant like.

5.2.4 Swarm Population



(a)



(b)

Graph 4: a. Field Defiance vs. Convergence Time in Firefly like Motion. b. in Ant like Motion

Number of Robots required from a fixed population is controlled by adjusting Robot Power. This Graph is in fact inverse of the previous Graph, with same properties.

Total Robot Population was 20.

5.3 Shape Extraction

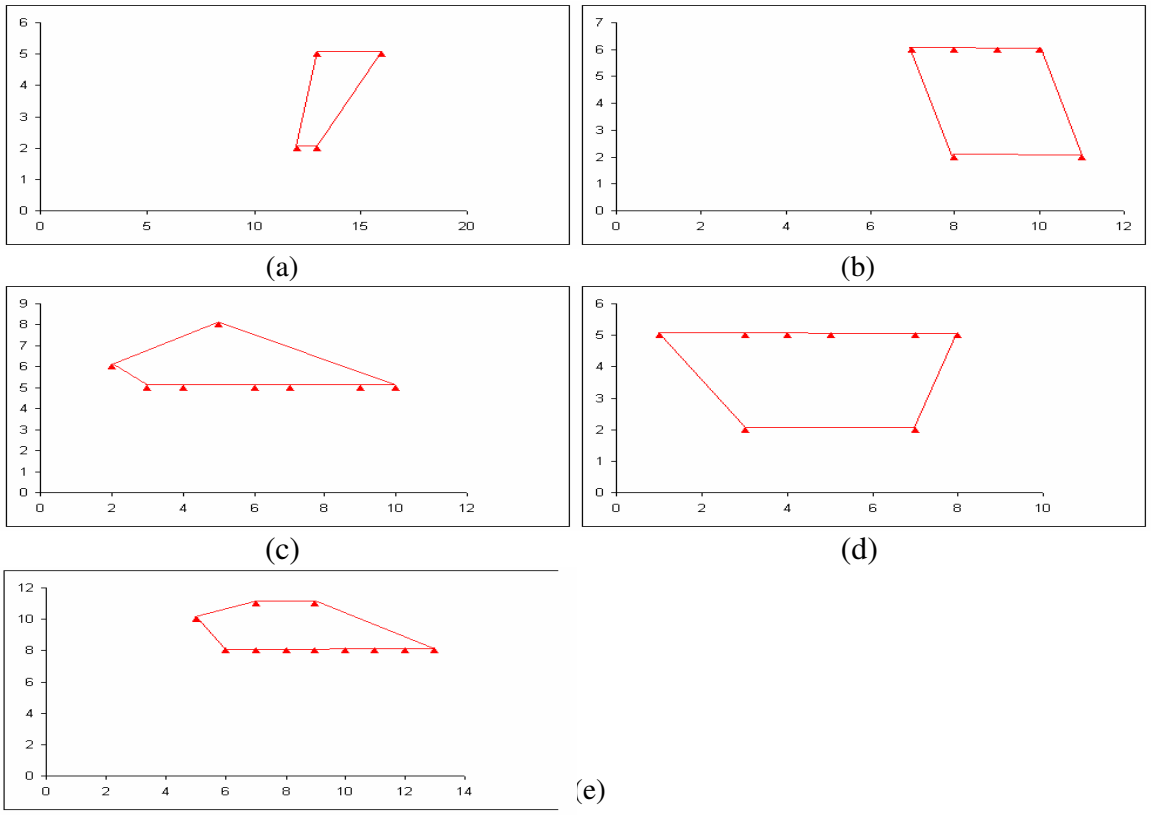


Figure 22: (a) 4 Robots give a quadrilateral. (b) 6 Robots show that the figure might be a Rhombus. (c) 8 Robots, any better ? (d) 8 Robots show that the object might be rectangular. (e) 11 Robots;a technique may be developed to combine different images to give a better view.

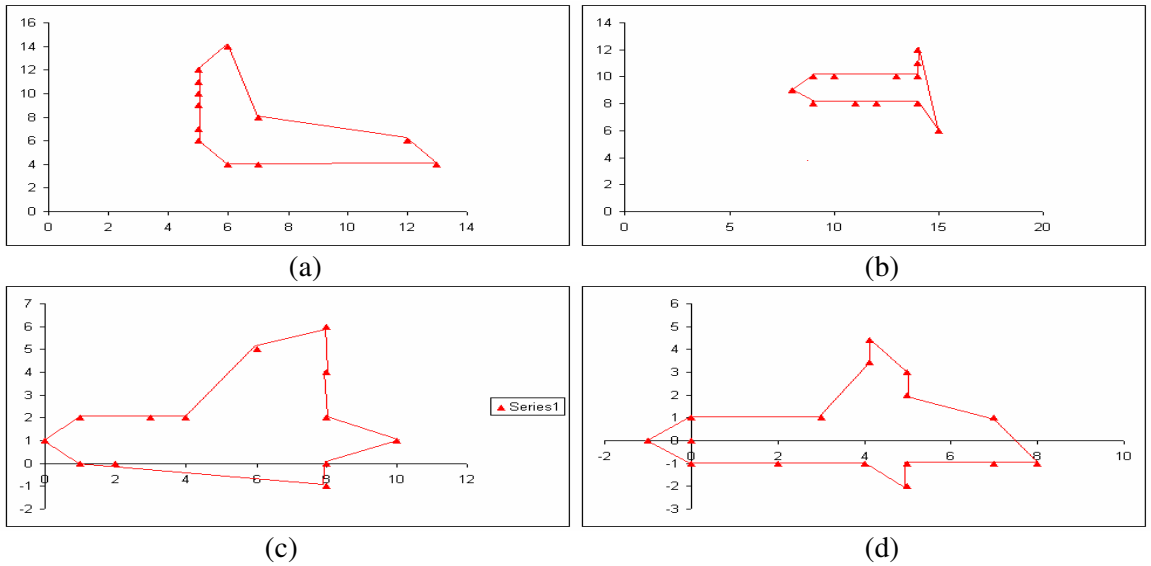


Figure 23: a. An 'L' shaped object with 12 Robots. b. '+' shaped object with 12 Robots c. '+' shaped object with 13 Robots d. '+' shaped object with 15 Robots that started with a better spatial distribution.

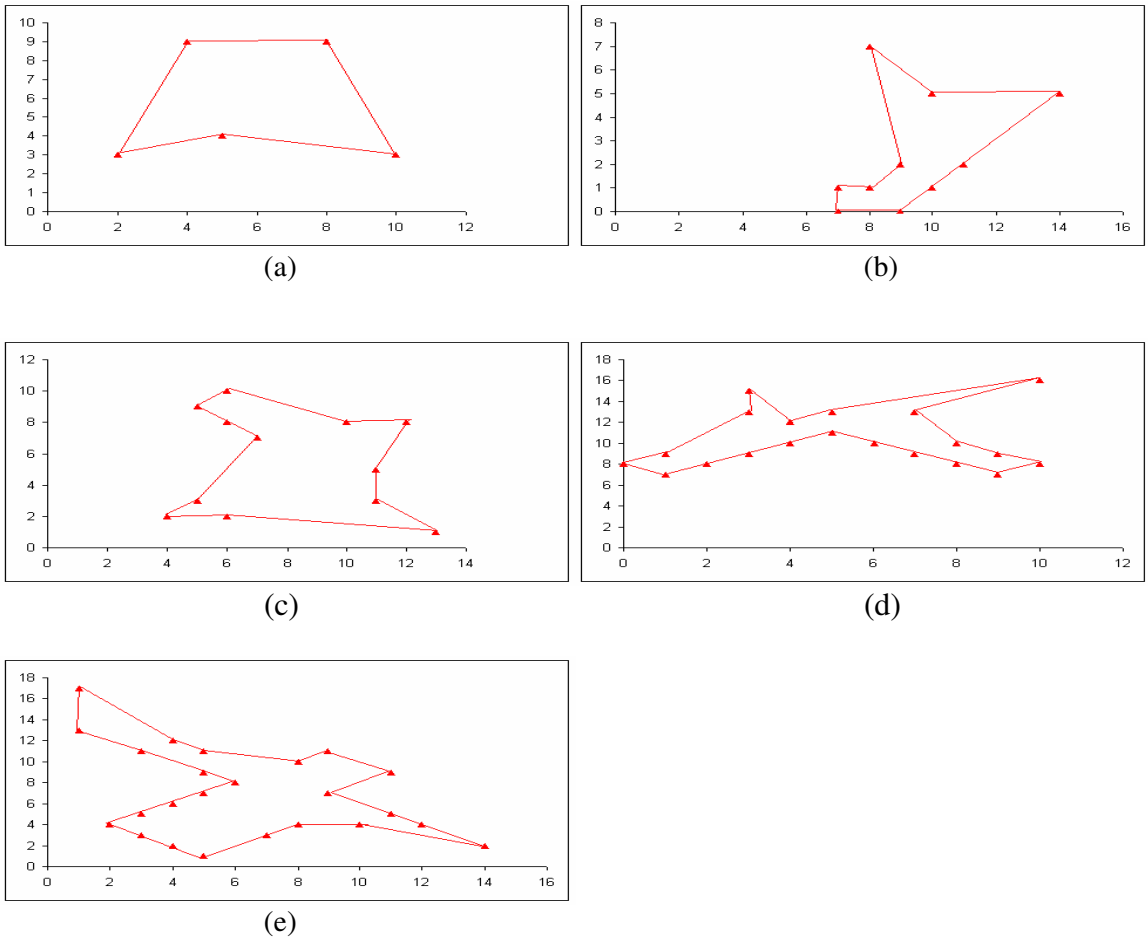


Figure 24: for an 'X' shaped object a. 4 Robots b. 10 Robots c. 12 Robots d. 20 Robots e. 24 Robots that started with a better spatial distribution.

5.4 Results

From the above graphs and figures, some conclusions can be made. In the graphs 1-4, it is obvious that Firefly like Robots generally converge quicker than Ant like Robots. Another fact can be seen in Graph 3, where 'powerful' Robots converge quickly as compared to 'weak' Robots. Signal radius is also found to be a significant factor, contributing positively to Swarm's convergence time.

Section 5.2, figures 22-24 show that larger the number of Robots, better the shape of object is extracted. Another factor found to be significant is spatial distribution of Robots. Wider the distribution of Robots, better they surround an object and hence better the shape of object is extracted. A factor not considered is the ratio between Robot size and object's surface area. In other words, it is the surface area that each Robot looks at and reserves for itself so that another Robot may not look at it (turns the patch blue). A swarm with low power robots reserving large surface areas might never converge. This draws attention to another factor, object density, and will be dealt with in an improved model.

A better algorithm to efficiently extract the shape from available coordinates is needed. In a future model, spatial distribution factor may also be introduced, so as to give a better degree of accuracy with respect to object recognition.

Chapter 6.
Conclusion and Future Work

Incremental Perception is the information built-up and shared by S*warm* of simple Robots (equipped with merely touch sensors and no visual device) to collectively recognize an object as their target. A hybrid Architecture has been adapted for the model in which the swarm starts with in a decentralised fashion. The model is heavily behaviour based where high-level behaviours arise from low level rules defined for Robots. The fact that makes this work unique is that there is no explicit communication between the Robots and the information being shared is only through the display of behaviours.

Each Robot displays a small set of simple yet consistent behaviours which gives rise to a complex colony. The agents wander around in the world looking for an object, unaware by the presence of other Robots. If, however, two Robots come close enough such that there is a possibility of collision between them, they turn around and adapt a different route. The model behaves in a Decentralised fashion until the first Robot finds an object.

Simulating Language NetLogo has been used whose advantages over other modelling languages have already been discussed. NetLogo is truly an ideal language for simulating Swarm Behaviour as it can cater agent colonies of thousands.

6.1 Improvements

By the time the present work came was completed, many improvements and extensions had already been identified, which considering the timescale of present project, were not possible to implement. Nevertheless, these will be rectified/added as future work. Some of these improvements are discussed below.

6.1.1 Field Defiance

While Robots move about within the potential field, fieldDefiance can make Ant like movement appear similar to Firefly like movement. In fact in the said conditions, they cannot be differentiated unless the model is run at 50% of full speed.
RandomHead is repeatedly called for a Robot moving within a field, the repetitions depending upon the value of fieldDefiance.

Sequence of instructions when Robot is inside a field

```
repeat fieldDefiance [ RandomHead ]
FindObject
if foundObject    [ stop ]
```

Table 14: call `RandomHead` fieldDefiance times and then try to find object. While this sequence introduces a randomness factor into movement of Robots inside a field, it affects Ant like movement of Robots.

`RandomHead` in turn calls `HeadCarefully`, which gives a Robot random heading if there is a possibility to collide with another Robot or a segment of an object that is already being looked at by another Robot. While within a field, Robots repeatedly confront other Robots and blue object segments, thus changing their heading randomly at a high rate that makes their movement firefly like even though they try to move like Ants.

The function `HeadCarefully` can be redesigned to force Ants to remain Ants. However it is anticipated that this will slow down the Ants movement inside field.

6.1.2 Behaviour of Robots when they attempt to bring object back

Before the Robots attempt to transport the object back home, they depend upon the leader to specify a direction to move in. While selecting the heading, the leader also alters direction of Robots that do not surround object, thus causing them to return. This may not always be the intention and can be eliminated by restructuring the code carefully. An external control may also be provided to take the decision.

6.1.3 Dynamic Field

The `found` variable can allow entry into `Signal` with coordinates of calling Robot as arguments. The field thus generated does not move around with its creating Robot. A different approach would be more efficient, in which a Robot itself creates a potential field around it. Preserving previous values of field in a given radius around the Robot can serve this purpose. Another approach in which a Robot only increments and decrements the field variable of patches around it can be more useful, but cannot be used in presence of diffuse field primitive, as there seems to be no control over the way that this primitive handles variables.

6.1.4 Obstacle Avoidance

Similar to Multiple Object Problem, it is considered that there are no obstacles to the movement of Robots. The presence of any obstacles, especially while the *swarm* is bringing an object back, becomes significant. This itself will be an arena to explore the performance of several obstacle avoidance algorithms.

6.1.5 Object Weight as a function of Density

objectWeight, at present, is an externally defined parameter and does not depend upon size of object. However, a realistic approach will be to carefully designate objectWeight depending upon an Object Density Function and Object Size.

$$objectWeight = \rho_{ext} \times \nu_{patch} \tag{4}$$

where ρ_{ext} is the density function whose value is specified externally, while ν_{patch} is the object size and is a function of number of patches that the object occupies.

Object weight should be replaced by *Object Density*. It is seen that there is a limit to the number of Robots that can surround an object, depending upon its area. If the power of maximum number of Robots that can surround an object does not equal or increase its weight, the *swarm* does not completely converge. In such a case shape may be extracted but *swarm* will not be able to bring the object back. Also, some of the Robots will continue to move with `foundField` set to true and never find an object.

6.2 Future work

6.2.1 Java Extension for an Efficient Controller

Shape extraction is external to the model at present. However, NetLogo provides facility of external Java plug-ins, a feature that can be exploited to generate an efficient controller. This controller will extract the shape of object and hence take the decision whether to bring the object back or not.

6.2.2 More Movement Models

Robots can only show to movement models namely Ant like and Firefly like. Other movements may be added to the model like Moth and Frog like and/or any other. Many other Formation and Marching behaviours may as well be introduced.

6.2.3 More than one Breeds of Robots

In the present model, Robots exist as one turtle breed and can behave either like Ants or Fireflies. A better approach would be to define two (or more) Robot breeds that can co-exist in the world and comply to a different set of rules.

6.2.4 Path Planning

While working with multiple objects, it would be essential to introduce path planning. Although a whole arena of study itself, known path planning techniques can be used to make the model more efficient.

6.2.5 Fuzzy Logic

A fuzzy logic based approach may be developed for collective perception of the shape of object by *swarm*.

6.2.6 Faults and Testing

To check system's scalability and robustness, faults may be introduced and the system performance tested against them. Some of the faults can be

- Death of some Robots
- Introduction of Dead Locks
- Some agents not complying to their behaviours
- Randomly moving agents

Appendix A

```
breed [robots robot]
breed [objcts objct]

robots-own
[ foundObject   foundField   isMobile    isLeader  ]
patches-own [ field    oldField    wayHome ]


globals
[
  found  ; first robot to find an object sets this variable to true
  locx    ; geographical coordinates of first robot when it senses
  locy    ; an object
  objX
  objY
  canTransport
  lHead
  power
  timeChek
  gCounter
  dra
]


to setup

  ca
  set found false
  set canTransport false
  CreateWorld
  CreateSwarm
  DrawObject
  reset-timer
  set timeChek true
  file-open "indstats.txt"
  file-print "New Sim"
  file-close
  set gCounter 0
  set dra true

end
```

```
to CreateSwarm

  create-custom-robots agents
  [
    set color red
    fd random max-pxcor
    set isMobile true
    set foundObject false
    set foundField false
    set isLeader false
    set size 1
  ]

end



to CreateWorld

  ask patches [ set pcolor white]
  set objx random 10
  set objy random 10
  DrawObject
  ask patches
  [ PaintHome ]

end



to PaintHome

    if distancexy -15 -15  < 2
    [
      set pcolor cyan
    ]
    set wayHome distancexy -15 –15

end
```

```
to go
  plot count robots with [ color != red ]
  ask robots
  [

    if isMobile
    [
      if not foundObject and not foundField
      [
         RandomHead
         FindObject
         if not foundObject
         [
           FindField
           ;set heading random 360
         ]
      ]

      if foundField and not foundObject
      [
         FollowField
         repeat fieldDefiance
         [
           RandomHead
         ]
         FindObject
         if foundObject [ stop ]
      ]

      if (not foundField) and foundObject
      [
         set field 10
         set isLeader true
      ]
    ]
  ]


  if ( found )
  [
     Signal locx locy
     set found false
  ]
  if not any? turtles with [ color = red]
  [
    file-open "shape.txt"
```

```
  ask turtles
  [
     if (foundObject)
     [
       file-write round xcor
       file-print round ycor
       ifelse isleader [ set color red ]
       [ set color green ]
     ]
  ]
  file-close
  set canTransport true
]

if power >= objectWeight
[

  if timeChek
  [
    file-open "ct.txt"
    ;file-write "CT"
    file-print timer
    set timeChek false
    file-close
  ]

  if dra [ DrawShape ]

  Transport

  ask turtles
  [
    set heading lHead
    fd 1
    if wayHome < 4
    [
      set canTransport false
    ]
  ]
  if any? robots with [ foundObject and distancexy -15 -15 < 1 ]
  [

    file-open "tt.txt"
    file-print timer
    file-close
```

```
    set gCounter gCounter + 1

    if not (gCounter = 50 )
    [
      setup
      go
    ]

  ]

  ask patches
  [
    set pcolor white
  ]

  ask patch-at objx objy
  [

      ask patch-at-heading-and-distance lHead 1
      [
      set objx pxcor
      set objy pycor
     ]

  ]

 ]

 ask patches [ PaintHome ]

 DrawObject

end
```

```
to DrawShape

  set dra false
  file-open "shape.txt"

    ask turtles
    [
       if (foundObject)
       [
         file-write round xcor
         file-print round ycor
         ifelse isleader [ set color red ]
         [ set color green ]
       ]
    ]

    file-close

end



to SetLocaion
    set locx xcor
    set locy ycor
end



to Signal [ tempx tempy ]
  ask patch-at tempx tempy
  [
    ask patches in-radius sigRadius [
      set field  ( field +  round (10 / (1 + distancexy tempx tempy)) )
    ]
  ]
  ask patch-at tempx tempy
  [
    ask patches in-radius sigRadius [
    ifelse ( (pcolor = blue) or ( pcolor = yellow ) ) []
    [ set pcolor scale-color red (10 * field) 90 10 ]
    ]
  ]
end
```

```
to Transport
  ask turtles
  [
    if isLeader
    [
      set color red
      set lHead downhill wayHome
    ]
  ]
end


to FindObject
  ifelse movementModel = 1
  [
    foreach [0 90 180 270]
    [
       if not foundObject
       [
         set heading ?
         LookForObject
       ]
       if not foundObject [ set heading random 360]
    ]
  ]
  [   Turn     LookForObject   ]
end



to LookForObject
  if pcolor-of patch-at-heading-and-distance heading 1 = yellow
  [
    set color yellow
    set pcolor-of patch-ahead 1 blue
    set foundObject true
    set isMobile false
    SetLocaion
    set found true
    set power power + rPower
    file-open "indstats.txt"
    file-print round timer
    file-close
  ]
end
```

```
to RandomHead

 ifelse movementModel = 1
 [    set heading random 360    HeadCarefully   ]
 [  HeadCarefully  ]

end


to HeadCarefully

 ifelse ( any? turtles-on ( patch-at-heading-and-distance heading 1) )
 or ( pcolor-of patch-at-heading-and-distance heading 1 = yellow)
 or ( pcolor-of patch-at-heading-and-distance heading 1 = blue)

 [     set heading random 360    ]
 [
   ifelse movementModel = 1
   [ fd 1 ]
   [ fd 0.25 ]
 ]

end


to Turn

 let fac ( random 100 mod 2 )
 ifelse fac = 0
 [ rt random robotVisionSpan ]
 [ lt random robotVisionSpan ]

end


to FindField

 if field-of patch-ahead 1 > 0
 [
   set foundField true
   FollowField
 ]

end
```

```
to FollowField
  set heading uphill field
  HeadCarefully
end


to DrawObject

  if object = 1
  [
    let a  1
    repeat 8
    [
      ifelse (pcolor-of patch-at (objX + a) objY  = blue)
     or (pcolor-of patch-at objX (objy + a) = blue)
     or (pcolor-of patch-at (objX + 1) (objy + a) = blue)  [ ]
      [
        set pcolor-of patch-at (objX + a) objY yellow
        set pcolor-of patch-at (objX + a) (objY + 1) yellow
      ]
      set  a ( a + 1 )
    ]
  ]

  if object = 2
  [
    let a  1
    repeat 8
    [
      ifelse (pcolor-of patch-at (objX + a) objY  = blue)
     or (pcolor-of patch-at objX (objy + a) = blue)
     or (pcolor-of patch-at (objX + 1) (objy + a) = blue)  [ ]
      [
        set pcolor-of patch-at (objX + a) objY yellow
        set pcolor-of patch-at (objX + 1) (objY + a) yellow
      ]
      set  a ( a + 1 )
    ]
  ]
if object = 3
  [
    let a  1
    repeat 8
    [
      ifelse ( pcolor-of patch-at (objX + a) objY  = blue)
     or ( pcolor-of patch-at (objX + 6) ( objY + 6 - a ) = blue)   []
```

```
    ;or (pcolor-of patch-at (objX + 1) (objy + a) = blue)  [ ]
      [
        set pcolor-of patch-at (objX ) objY yellow
        set pcolor-of patch-at (objX + a) objY yellow
        set pcolor-of patch-at (objX + 6) ( objY + 6 - a ) yellow
      ]
      set  a ( a + 1 )
    ]
  ]

if object = 4
  [
    let a  1
    repeat 8
    [
      ifelse (pcolor-of patch-at (objX + a) (objY + a ) = blue ) []
     ; = blue) or (pcolor-of patch-at objX (objy + a) = blue)
     ;or (pcolor-of patch-at (objX + 1) (objy + a) = blue)  [ ]
      [
        set pcolor-of patch-at (objX ) (objY ) yellow
        set pcolor-of patch-at (objX + a) (objY + a ) yellow
        set pcolor-of patch-at (objX + a) (objY + 8 - a ) yellow
      ]
      set  a ( a + 1 )
    ]
  ]
end
```

References

1. Beni, G., "The Concept of Cellular Robotic Systems", Proc. 3rd IEEE Int'l symp. Intelligent Control, Arlington, VA, pages 57-62, August 24-26, 1988

2. H. Van Dyke Parunak, "Making *Swarm*ing Happen", Presented at the Conference on Swarm and C4ISR, Tysons Corner, VA. 3, Jan 2003

3. S. Hackwood and G. Beni., "Self-organizing sensors by deterministic annealing", In IEEE / RSJ IROS, pages 1177-1183, 1991

4. E. Bonabeau, M. Dorigo, and G. Theraulaz., "Swarm Intelligence: From Natural to Artificial Systems", New York, Oxford University Press, 1999.

5. H. Wedde and M. Farooq., "The wisdom of the hive applied to mobile ad-hoc networks", Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE, Page(s): 341 – 348, 8-10, June 2005

6. C. Cianci, V. Trifa, A. Martinoli., "Threshold-based algorithms for power-aware load balancing in sensor networks", In Proceedings of The IEEE Swarm Intelligence Symposium SIS-2005, Pasadena, California, June, 2005.

7. R. Montemanni and L.M. Gambardella., "Swarm approach for a connectivity problem in wireless networks", Proceedings of the IEEE Swarm Intelligence Symposium (SIS 2005), pages 265-272, Pasadena, U.S.A., 8-10, June 2005.

8. Wan, Y. Sandip Roy Saberi, A. Lesieutre, B., "A stochastic automaton-based algorithm for flexible and distributed network partitioning", Swarm Intelligence Symposium, 2005. SIS-2005. Proceedings 2005 IEEE, On page(s): 273- 280, 8-10 June 2005.

9. Klein, J., "Continuous 3D Agent-Based Simulations in the breve Simulation Environment". In Proceedings of NAACSOS Conference (North American Association for Computational, Social, and Organizational Sciences). Pittsburgh, PA, 2003

10. Klein J., "breve: a 3d Simulation Environment for Multi-Agent Simulations and Artificial Life", breve Documentation version 2.4.

11. Spector, L., Klein J., and Keijzer M., "The Push3 Execution Stack and the Evolution of Control". In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2005), pp. 1689-1696. Springer-Verlag, 2005.

12. NetLogo itself: Wilensky, U., NetLogo. http://ccl.northwestern.edu/NetLogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL. 1999.

13. NetLogo documentation: Wilensky, U. Frequently asked questions. http://ccl.northwestern.edu/NetLogo/faq.html/. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL, 1999.

14. Wilensky, U. & Stroup, W., HubNet. http://ccl.northwestern.edu/NetLogo/hubnet.html. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL, 1999.

15. G. Beni and U. Wang., "Swarm intelligence in cellular robotic systems". In NATO Advanced Workshop on Robots and Biological Systems, Il Ciocco, Tuscany, Italy, 1989.

16. Tuytelaars, T., Zaatri, A., Van Brussel, H. and Van Gool, L., "An object recognition as part of a supervisory control system", in ICRA'2000, San Francisco, 2000.

17. Dorf R., Concise International Encyclopedia of Robotics: "Applications and automation". Wiley-Interscience, 1990.

18. Theraulaz G., Bonabeau E., "Coordination in distributed building," Science 269, 686-688, 4 August 1995.

19. Theraulaz G., Bonabeau E., J.L. Deneubourg, "The algorithmic beauty of stigmergic patterns in social insects," Math. Sci. 435:52-65, 1999.

20. Baldassarre, Nolfi S., Parisi D., "Evolving Mobile Robots Able to Display Collective Behaviors". Proceedings of the International Workshop on Self-Organizing and Evolution of Social Behavior, pages 11-22, Ascona, Switzerland, September, 2002.

21. Martinoli A., "Swarm Intelligence in Autonomous Collective Robotics: From Tools to the Analysis and Synthesis of Distributed Control Strategies". Ph.D Thesis Nr. 2069.

22. C Reynolds., "An evolved, vision-based model of obstacle avoidance behavior", in C. Langton, editor Artificial Life III, Santa Fe Institute Studies in the Sciences of Complexity, proc. Vol. XVI Wesley, 1993.

23. Werner, G. M. and Dyer, M. G., "Evolution of Herding Behavior in Artificial Animals". In: J.-A. Meyer, H. L. Roitblat, and S. W. Wilson (eds.), *From Animals to Animats 2*: *Proceedings of the Second International Conference on Simulation of Adaptive Behavior.* Bradford Book?MIT PRess, Cambridge MA. pp. 393-399, 1993.

24. Ward, C. R., Gobet, F., & Kendall, G., "Evolving collective behavior in an artificial ecology". Artificial Life, 7, 191-209, 2001.

25. Parker L. E., “Alliance; an architecture for fault tolerant, cooperative control of heterogeneous mobile *Robot*s”. In IEEE/RSJ IROS, pages 776-783, 1994.

26. Parker L. E., “Hetrogeneous Multi-Robot Cooperation. PhD thesis”, MIT EECS Dept., February 1994.

27. Y. Uny Cao, Alex S. Fukunaga , Andrew B. Kahng, “Cooperative Mobile Robotics: Antecedents and Directions”, 1997.

28. Wessnitzer J., Adamatzky A., Melhuish C., “Towards Self-Organising Structure Formations: A Decentralized Approach”. 573-581; ECAL 2001.

29. G. Beni and J. Wang, "Theoretical Problems for the realization of distributed *Robot*ic systems", Proceedings of the 1991 IEEE International Conference on Robotics and Automation, 1991.

30. Melhuish C., Holland O. and Hoddell S., "Collective sorting and segregation in *Robot*s with minimal sensing", 5th International Conference on the Simulation of Adaptive Behaviour, From Animals to Animats, MIT Press, 1998.

31. McFarland D., “Towards Robot cooperation”, In Proc. Simulation of Adaptive Behavior, 1994.

32. Erkin Bahceci, Onur Soysal, Erol Sahin, “A Review: Pattern Formation and Adaptation in Multi-Robot Systems”, Robotics Institute Carnegie Mellon University Pittsburgh, Pennsylvania 15213 (2003).

33. Egerstedt M. and Hu X., “Formation constrained multi-agent control”. IEEE Transactions on Robotics and Automation, 17(6):947.951, 2001.

34. Koo T.J. and Shahruz S.M., “Formation of a group of unmanned aerial vehicles (uavs)”. In Proceedings of the American Control Conference, 2001.

35. Kowalczyk W., “Target assignment strategy for scattered *Robot*s building formation”, in RoMoCo'02. Proceedings of the Third International Workshop on Robot Motion and Control, 2002.

36. Belta C. and Kumar V., “Trajectory design for formations of *Robot*s by kinetic energy shaping”, In Proceedings. ICRA'02. IEEE International Conference on Robotics and Automation, 2002.

37. Erkin Bahceci, Onur Soysal and Erol Sahin, “A Review: Pattern Formation and Adaptation in Multi-Robot Systems”, CMU-RI-TR-03-43 October 2003.

38. Mataric M., “Designing emergent behaviors: From local interactions to collective intelligence”, in Proceedings of the International Conference on Simulation of Adaptive Behavior: From Animals to Animats 2, pages 432-441, 1992.

39. Mataric M., “Minimizing complexity in controlling a mobile *Robot* population”, in proceedings of the 1992 IEEE International Conference on Robotics and Automation, pages 830-835, Nice, France, May 1992.
40. Albert L. Schoute, “Time-optimal collision avoidance of automatically guided vehicles”, University of Twente, Department of Computer Science Postbox 217, 7500AE Enschede, Netherlands.
41. James D. McLurk, “Stupid Robot Tricks: A Behavior-Based Distributed Algorithm Library for Programming Swarms of Robots”, Massachusetts Institute of Technology, May 2004.
42. Lerman, K. and Galstyan, “A. Mathematical model of foraging in a group of robots: Effect of interference”, Autonomous Robots, 13(2):127–141, 2002.
43. Tobias, R., Hoffmann, C, “Evaluation of the Free Java Libraries for Social scientific Agent Based Simulation / Internet.” - http://jasss.soc.surrey.ac.uk/7/1/6.html in Journal of Artificial Societies and Simulation
44. AN OVERVIEW OF THE AGENT - BASED SOCIAL SYSTEM SIMULATION TOOLS Dmitrij Pozdnyakov, Riga Technical University 1, Kalku Street, Riga, LV-1658, Latvia, Annual Proceedings of Vidzeme University College “ICTE in Regional Development”, 2006

Bibliography

Following works, although not cited directly, have been very helpful during my work.

K. Nickels, A. Castano, C. Cianci., "Fusion of Lidar and Stereo Range for Mobile Robots". In Proceedings of The 11th International Conference on Advanced Robotics, Coimbra, Portugal, July 2003.

R. Montemanni, J. Barta and L.M. Gambardella**. "**An exact algorithm for the robust traveling salesman problem with interval data". Proceedings of ODYSSEUS 2006, pages 256-258, Altea, Spain, 23-26 May 2006.

A.E. Rizzoli, N. Casagrande, A.V. Donati, L.M. Gambardella, D. Lepori, R. Montemanni, P. Pina and M. Zaffalon. "Planning and optimisation of vehicle routes for fuel oil distribution". Proceedings of MODSIM 2003 - Integrative Modelling of Biophysical, Social and Economic Systems for Resource Management Solutions February, volume 4 pages 2024-2029, Townsville, Australia, 11-17 July 2003.

R. Montemanni , J. Barta and L.M. Gambardella., "Heuristic and preprocessing techniques for the robust traveling salesman problem with interval data". Technical Report IDSIA-01-06, Istituto Dalle Molle di Studi sull'Intelligenza Artificiale, January 2006.

Crawford-Marks, R., and L. Spector. "Size Control via Size Fair Genetic Operators in the PushGP Genetic Programming System". In W. B. Langdon, E. Cantu-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska (editors), Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2002, pp. 733-739. San Francisco, CA: Morgan Kaufmann Publishers. 2002.

Spector, L., and A. Robinson., "Genetic Programming and Autoconstructive Evolution with the Push Programming Language". In Genetic Programming and Evolvable Machines, Vol. 3, No. 1, pp. 7-40. 2002

Bonabeau E., Dorigo , and Theraulaz G., "From Natural to Artifcial Systems". Oxford University Press, 1999.

Martinoli, A. J. Ijspeert, and L. G. Gambardella, "A Probabilistic Model for Understanding and Comparing Collective Aggregation Mechanisms', Proc. of the Fifth Int. European Conf. on Artificial Life ECAL-99, Lausanne, Switzerland, pp. 575-584, September 1999.

Zaera, N., Cliff, D., & Bruten, J., "(Not) evolving collective behaviours in synthetic fish". Technical Report HPL-96-04. Palo Alto, Ca.: Hewlett-Packard Laboratories. 1996.

Seeley, Thomas D. “The Wisdom of the Hive” Harvard University Press. Cambridge, Massachusetts. 1995.

P.K.C. Wang. “Navigation strategies for multiple autonomous Robots moving in formation”. Journal of Robotic Systems, 8(2):177:195, 1991.

Q. Chen and J. Y. S. Luh., “Coordination and control of a group of small mobile Robots”. In Proceedings of the 1994 IEEE International Conference on Robotics and Automation, pages 2315{ 2320, San Diego, CA, USA, 1994.

Martinoli A., Easton K. and Agassounon W., “Modeling of Swarm Robotic Systems: A Case Study in Collaborative Distributed Manipulation”. Special Issue on Experimental Robotics, Siciliano, B., editor, Int. Journal of Robotics Research, 23(4): 415–436, 2004.

Image References

Figure 1a. Image Source: www.sandia.gov/media/minebees.htm, "Bees foraging near a sugar-water feeder"; 'bees.jpg';
Figure 1b. Image Source: www.thesahara.net/ants.jpg, "Ants surround a toxic gel", 'ants.jpg'
Figure 2. Image Source: http://www.k-team.com/kteam/home.php, Khapera II.
Figure 3. Image Source: Image source: FiBreve Documentation: version 2.4 Chapter 14: 'The breve Source Code', The breve software architecture
Figure 4a. Image Source: http://www.nps.gov/archive/dena/home/resources/Wildlife/ birdweb/ index/ birdwatchTS.htm
Figure 4b. http://www.math.utah.edu/~kfitzger/research.html

Some Useful URLs

Home page of Prof. Roberto Montemanni
http://www.idsia.ch/~roberto/

Home page of Christopher Cianci, Swarm Intelligence Systems Group
http://www5.epfl.ch/swis/page1339.html

Home page of Lee Spector, Hampshire College
http://hampshire.edu/~lasCCS/publications.html

John Klein Publications
http://artificial.com/publications.html

Lee Spector, Publications
http://hampshire.edu/~lasCCS/publications.html

Kinematic Self-replicating Machines
http://www.molecularassembler.com/KSRM/Refs2700-2799.htm

Home page of Guy Theraulaz
http://cognition.ups-tlse.fr/_guyt/

Molecular Assembled Website.
http://www.molecularassembler.com/index.htm

Computational and Mathematical Organization Theory, 5(3), October, Special Issue on Social Intelligence; Edited by Bruce Edmonds and Kerstin Dautenhahn
Dordrecht: Kluwer Academic Publishers 1999; ISSN 1381-298X
http://jasss.soc.surrey.ac.uk/5/3/reviews/gotts.html#theraulaz1999

Prof. Michael G. Dyer, Computer Science Department, 4532F Boelter Hall, University of California at Los Angeles (UCLA), Los Angeles, CA 90095-1596
http://www.cs.ucla.edu/~dyer/Publications.html