University of Southern Queensland
School of Mechanical and Electrical Engineering

A General Architecture for Robotic Swarms

A thesis submitted by

Iain James Brookshaw

in fulfilment of the requirements of

Doctor of Philosophy

2015

Abstract

Swarms are large groups of simplistic individuals that collectively solve disproportionately complex tasks. Individual swarm agents are limited in perception, mechanically simple, have no global knowledge and are cheap, disposable and fallible. They rely exclusively on local observations and local communications. A swarm has no centralised control.

These features are typified by eusocial insects such as ants and termites, who construct nests, forage and build complex societies comprised of primitive agents.

This project created the basis of a general swarm architecture for the control of insect-like robots. The Swarm Architecture is inspired by threshold models of insect behaviour and attempts to capture the salient features of the hive in a closely defined computer program that is hardware agnostic, swarm size indifferent and intended to be applicable to a wide range of swarm tasks.

This was achieved by exploiting the inherent limitations of swarm agents. Individual insects were modelled as a machine capable only of perception, locomotion and manipulation. This approximation reduced behaviour primitives to a fixed tractable number and abstracted sensor interpretation. Cooperation was achieved through stigmergy and decisions made via a behaviour threshold model.

The Architecture represents an advance on previous robotic swarms in its generality – swarm control software has often been tied to one task and robot configuration. The Architecture's exclusive focus on swarms, sets it apart from existing general cooperative systems, which are not usually explicitly swarm orientated.

The Architecture was implemented successfully on both simulated and real-world swarms.

Certification of Dissertation

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.
I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Iain James Brookshaw

w0086292

_____

Signature of Candidate

_____

Date

Endorsement

_____

Signature of Supervisor/s

_____

Date

Acknowledgements

Many people helped make this project possible, but I would like especially to thank my father, Dr. Leigh Brookshaw, whose knowledge of Linux made this project possible, my mother Jan for her tireless support and critical proofreading (without which this document would be close to illegible) and my sister Ingrid, who knows how to fix a cow.

I would also like to thank my friends from school and uni: Smithy, Robin, Cameron, Erin, Sabrina, Chris J, Rian, Luke, James, Rodolfo, Tiarna, Safeen, Tanya, Scanlon, all those who wrote for the paper (despite everything), everyone from C-block... and many, many others, too numerous to name and in no order. You all know why and who you are – thank you.

I would also like to acknowledge all those thousands who have laboured to create open-source hardware and software, without which almost no robotics research would be possible and more locally to thank Richard, Les and Selvan who made important technical suggestions and Dean, Graham, Terry and Adrian who know how to get things done and where everything is.

Finally and most importantly, I would like to thank Dr. Tobias Low, who has helped, contributed and put up with my work for the last four years, been consistent with advice, selfless with help and who always knows what to do next.

THANK YOU

Colophon

This document was typeset in Computer Modern by the LaTeX2e typesetting program.

The epigraphs at the start of each chapter are quotations from *The Hitchhiker's Guide To The Galaxy* by Douglas Adams, as broadcast on BBC Radio in 1978.

This work was made possible through an Australian Government Postgraduate Award Scholarship.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

> In the beginning the universe was created. This has made a lot of people very angry and was widely regarded as a bad move.
>
> — Douglas Adams,
> *The Hitchhiker's Guide to the Galaxy*

Contents

A SWARM IS A LARGE, but arbitrarily sized group of autonomous individuals that cooperate to solve relatively complex tasks as a group. These tasks are often so much larger than the individuals and cooperation so fluid and responsive that external observers perceive the illusion of planned, centralised control. In actuality, a swarm is comprised of extremely simple individual "agents" that coordinate via local interactions only.

The philosophy behind an artificial swarm is that it is better to build a large number of less-than-perfect simple, inexpensive robots, than a small number of highly complex, costly and sophisticated robots. This argument is supported by the activities of eusocial[1] insects – such as ants and termites – who build complex societies with relatively primitive agents.

[1] "truly social insects" where the vast majority of the colony have no reproductive presence and are sacrificed for the good of the colony at need(Hölldobler & Wilson 1995)

1.1 Objectives

This project aimed to develop the basis for a general software architecture for robotic swarm agents, inspired by eusocial insects. Currently, insect-like robotic swarm agents are controlled by software that is tightly focused on a specific task, while general robot architectures are not usually focused upon insect-like swarms (compare Sections 2.2 and 2.3 with Sections 2.5 and 2.6 respectively).

This project aims to rectify both of these problems, ultimately creating a system architecture that would form the basis of a multi-caste swarm of insect-like robots.

To achieve this, the architecture design was required to be:

- Applicable to many different insect-like swarm robot designs without major modification.
- Applicable to multiple different insect-like swarm tasks without major modification.
- Implemented in a fashion consistent with insect-like swarms. This implied:
  - Minimal communications, with coordination based upon agent observations. Insect-like methods of observation-based self-organisation are discussed in Section 2.3.
  - Fully autonomous agents and fully distributed swarms, no centralised control or localisation systems (see Section 2.3).
  - Minimalist agents, with as few moving parts as possible (see Section 2.1).

To achieve these objectives swarm-insect cooperation was examined in detail:

- The expected physical form of a insect-like swarm agent is exploited to sharply limit the scope and number of behavioural primitives (see Sections 2.1 and 3.3).
- Insect cooperation mechanisms and tasks were discussed (see Sections 2.2 and 2.3) and incorporated into the system design (see Sections 3.1 and 3.2).
- A model for eusocial insect behaviour was incorporated into the design as the basis for agent decision making (see Sections 2.4 and 3.4).

This project aimed to implement this architecture as a proof-of-concept only, demonstrating basic functionality and feasibility on idealised insect-like swarm tasks. Producing the "fastest" or "most efficient" swarm architecture is beyond the scope of this project.

See Chapter 5 for a discussion on future implementations.

Upon implementation, the architecture was found to be functional on several, idealised swarm tasks and two distinct swarm agents (one real-world, one simulated). The simulated swarm performed more smoothly, while the real-world swarm supported the simulated and emphasised key points for implementation, especially the need for unambiguous visual identification of object types.

1.2 Swarms

> "Swarm robotics is the study of how large numbers of relatively simple physically embodied agents can be designed such that a desired collective behaviour emerges from the local interactions among agents and between the agents and the environment" (Şahin 2005)

Swarms are always comprised of simple autonomous machines. They have no central controller, no hierarchy and no leader, elected or otherwise. The swarm cooperates without a master plan or global knowledge.

Instead, swarm cooperation is the result of agent-to-agent and agent-to-environment interactions. Agents are reactive, responding only to what they perceive in their immediate environment. Agents do not issue commands or share detailed information. It is the Olympian perspective of the human observer generates the illusion of centralised control. In Chapter 2 we discuss both insect colony projects and how this implicit coordination is achieved.

The simplicity of swarm agents is also emphasised in this chapter. In this project, we approximate worker insects as machines that can perceive their environment (classify objects and identify stimulus), move through the environment (from point A to B, although the means of locomotion is not considered important) and manipulate objects that they find there.

We assume that in individual agents all of these features are limited; perception is local and small-scale, locomotion is probably slow while manipulation is limited to a single, agent-scale object at a time.

We also expect that swarm agents are mechanically imperfect, it is the summed effect of the swarm that is important, not the actions of individuals. It is to be expected that individual swarm agents break-down, fail in their task and cannot grasp "the big picture", we assume that another agent is always present to help.

Efficiency in agent numbers, time or resources is not the objective, it is sufficient that the swarm achieves its task. We do not attempt to optimise cooperation.

These factors are summarised in Brambilla, Ferrante, Birattari & Dorigo (2013) and form what we shall term **the swarm ethos**:

- Robots are *autonomous* – see also Baldassarre et al. (2007), Şahin (2005).
- Robots are *situated* in the environment and can act to modify it – ie: they "do not deal with abstract descriptions, but through its sensors with the here and now of the world, which directly influences [its] behaviour" (Brooks 2002).
- Robots' *sensing* and *communication* are local – see also Liu, Winfield, Sa, Chen & Dou (2007), Nouyan et al. (2009). These features are heavily stressed and leads to agents with limited cognitive scope and low mechanical complexity – see Schmickl, Möslinger & Crailsheim (2007), Labella, Dorigo & Deneubourg (2006), Liu et al. (2007).
- Robots do not have access to *centralised control* and/or to *global knowledge*.

- Robots *cooperate* to tackle a given task. Moreover, individual swarm robots should be "*relatively* incapable or inefficient on their own with respect to the task at hand." (Şahin 2005). The task given to these localised robots is usually much greater in scale than they are and can only be completed through the summation of swarm actions – see Phan & Russell (2012), Nouyan et al. (2009).

  Instead of complexity, large numbers of agents are employed. Swarm robotics projects often attempt to aim for scalability to arbitrarily large numbers of robots (Şahin 2005, Werfel, Bar-Yam, Rus & Nagpal 2006, Garnier, Jost, Gautrais, Asadpour, Caprari, Jeanson, Grimal & Theraulaz 2008, Martinoli, Easton & Agassounon 2004, Rubenstein, Ahler & Nagpal 2012).

1.3 Swarm Tasks

Chapter 2 describes several selected swarm insect tasks. These are tasks which are suitable for swarm cooperation, where self-organised coordination emerges implicitly from agent interactions, large structures are built from repeating local units and task execution order is implicit in the environment.

These tasks may be undertaken in a dangerous, dynamic environment, spread over a large region. They may also scale arbitrarily (Şahin 2005). However, swarm tasks are also robust to individual, local failure. It is the sum of all agent's efforts that is important.

Tasks that require extreme precision, tightly coupled interaction and are intolerant to agent mistakes (ie: require verifiable levels of individual precision, accuracy and efficiency) are not considered swarm tasks.

1.4 The Architecture

Robotic swarms and other styles of cooperative groups have been extensively employed to solve a variety of tasks. Machine collectives that map, follow, explore, manipulate objects and build structures have been successfully implemented. Groups of robots have been used to explore control theories and examine or explicitly model insect behaviour. However, many of these projects are explicitly focused on solving the one task and their software architecture and behaviour structure is written, from a very low level, around that task exclusively.

To overcome this restriction, several general cooperative architectures have been developed. Usually behaviour-based, these architectures attempt to describe the perception and action process in a more general fashion allowing for the group to be assigned different tasks by changing the individual agents' behaviour patterns.

While successful in a variety of tasks, previous general architectures often achieve global, group wide coordination by the imposition of restrictions that are counter to the swarm ethos. As they are unable to exploit known task or environmental considerations, general architectures often require that robots explicitly share information along tightly defined channels. They also aim for robots that are too complex to be called swarm agents, attempting to control groups of complex robots, rather than swarms of simple machines.

This project seeks to overcome both these restrictions and designs a system that is focused exclusively upon swarms, but is sufficiently abstracted from the hardware to allow for different robot designs and task applications.

This project is a proof-of-concept, we do not intend to outperform existing, focused swarms at this time. Instead we aim to create a general system that draws on the key points of swarm cooperation, with the capacity for expansion.

1.4.1 What it is...

The Architecture follows the following broad principles:

Biologically Inspired:
The cooperative strategies of eusocial insects are adapted as a starting point. However, this project does not attempt to slavishly reproduce insects in robots, insect components are used where useful.

Hardware Agnostic:
A swarm agent is assumed to have rich local sensors, able to move through its environment and manipulate single objects. The specific hardware mechanisms used to do this are irrelevant. The Architecture does not care if the agent uses wheels, treads or flies, so long as it moves. In this way, The Architecture is **portable**.

It should be stressed that only swarm-like robots are supported, a swarm-like machine should use the bare minimum of moving parts. By examining swarm insects, we shall closely define a swarm agent as a machine capable of locomotion, simple manipulation and perception. Complex robots with many effectors and degrees of freedom are not considered swarm-like.

Task Agnostic:
The Architecture is based on the insects discussed in Chapter 2, rather than on any one task. The Architecture does not exploit *a priori* task knowledge, other than the assumption that the task will be "swarm-like" (see above). The same decision-making program is intended to be used for all tasks, regardless of goals.

Swarm Size Indifference:
The architecture was intended to be unrelated to swarm size, the design objective required that individual agents pursue their goals regardless of whether they are solitary or part of a swarm of millions.

It is important to note that these are the design principles of The Architecture. The system is only tested for broad functionality – to see if a system built on these principles can solve simple tasks and produce cooperation. This is discussed further in Chapter 4.

The following fields are rigorously defined by The Architecture:

- Basic motor schema or atomic behaviours (here termed "actions"). Both the number and objectives of these functions are tightly controlled and based on swarm considerations.
- Stigmergic, swarm orientated procedure for the creation of user-defined behaviours from these atomic components.

- User-defined behaviour input/output.
- A thin universal sensor abstraction.
- Minimum sensor, motor and swarm agent physical requirements.

The design is discussed fully in Chapter 3.

1.4.2 . . . and What it is Not

The Architecture not intended to be the "best", "fastest" or "most efficient" or "most capable" cooperative robot design. These concepts are not considered, it is only intended to control swarms. Cooperation among these very limited agents is the end, in and of itself.

In the same vein, The Architecture is not intended to serve every conceivable robot design. It only supports insect-like swarm agents and tasks. The robots are insect-like, they are not perfect and will make mistakes as individuals.

The following areas are beyond the scope of The Architecture:

- Sensor data pre-processing. A specific final result is required, but the intermediary steps are unimportant.
- Motor control. Specific results are required and certain functions must be filled, but control implementation is unimportant.
- Communications. In the current version of the architecture all agents are mute and deaf. For full generalisation, even local communication is eschewed (see Sections 3.2.3 and 5.1.2 for more details).
- Precise agent-to-agent interactions. Tasks are accomplished by the sum of interactions across the swarm as a whole, not by detailed or complex interactions between individuals.
- Temporal sequence. Timing is implicit in the behaviour construction and logical conditions. Beyond a certain level, the user may not expressly specify task execution order.
- Biological duplication. The Architecture design was not intended as a direct reproduction of biological systems. Insect swarms were taken as an inspiration and a starting place, not an end.

1.4.3 Implementation

We test The Architecture on simple idealised swarm tasks (see Chapter 4). These experiments are intended to show that The Architecture is functional and illustrate the practicalities of implementation.

The Architecture is implemented upon two swarms, one real and one simulated. The real-world robots are less robust then their simulated counterparts and are used to show that the simulations are effective. The real-world robots were designed to be as minimalist as possible – a process that went through several revisions, see Appendix B. They are intended to showcase The Architecture within the real-world constraints of limited time and a small budget.

The experiments are not intended to exhaustively test The Architecture, simply to show that it functions in a swarm-like fashion and provides a basis for further work. We discuss how The Architecture can be expanded, rendered more robust and deployed on practical problems in Chapter 5.

Chapter 2

Swarms: Insects and Robots

"... It's imprinted on the Earth man's brain wave pattern, but I don't suppose you'd be interested."

"You mean you can see inside my mind?"

"Yes."

"... and?"

"It amazes me that you can live in anything so small."

— Douglas Adams,
The Hitchhiker's Guide to the Galaxy

Summary

In this chapter, we shall discuss both insect and robotic swarms. This discussion will emphasise the key features of insect agents, both a individuals and as a group. As many of these features are counter-intuitive, they are discussed in detail to provide background to the architecture design, as well as the limitations of insect-like agents and appropriate task scope.

Contents

It is common for artificial swarms to be inspired by biological precedents: fish (Min & Wang 2010) and birds (Thalmann & Musse 2007) have both been used as models. However, by far the most common source of inspiration is "... the observation of social insects – ants, termites, wasps and bees – which stand as fascinating examples of how a large number of simple individuals can interact to create collectively intelligent systems" (Şahin 2005). Despite their simplicity as individuals, eusocial insects "... colonies are consistently superior to solitary and pre-eusocial competitors, due to the altruistic behaviour among nest-mates and their ability to organise coordinated action..."

Eusocial insects "represent the largest research corpus... [and] their underlying principles are very close to those found in other animal species" (Garnier, Gautrais & Theraulaz 2007). This project focuses almost exclusively on ants and termites, as bees are "considered the most highly evolved of all social insects" (McGavin 2001) and exhibit considerably more complicated communication patterns than their grounded counterparts. We shall begin at a simpler level.

Wilson & Hölldobler (2005) outline some of the principle features of eusocial insects, emphasising the existence of a distinct reproductive caste is attended by large numbers of non-reproductive workers and drones. Indeed, the vast majority of individuals in a colony have no offspring (and consequently have no evolutionary presence) and are often sacrificed for the colony's survival (Drickamer, Vessey & Jakob 1996).

Although swarm robots are not expected to reproduce, this is a critical feature for swarm robot development; individual workers are interchangeable and expendable. The loss or failure of a percentage of worker agents is not significant. From this it follows that insect-like swarm robots should be as simple and inexpensive as possible, even at a cost in individual performance.

2.1 Insects As Machines

To realise this in robots, we roughly approximate individual insects as simple machines subdivided into two parts: locomotion/manipulation and perception/-communication.

Perception and Communication Ants and termites have localised perception and communication channels. Broadly, both species have compound eyes (Sudd 1970) (Howse 1970), although their vision is usually poor (Gordon 1999) (Sudd 1970). They are also sensitive to tactile input (Turner 2011) (Gordon 1999) and may be sensitive to other locally perceived environmental factors, such as humidity, carbon-dioxide concentration or vibration (Turner 2011).

However, both species rely heavily upon "rich medium of chemical communication between [agents], mediated mostly by pheromones." (Turner 2011), (Gullan & Cranston 1994). This sensing modality is "the glue that binds the complex but highly ordered world of the ant colony... the most pervasive mode of communications..." (Hölldobler & Wilson 1989). Indeed, ants "perceive the world, and each other, mostly though odours" (Gordon 1999).

Pheromones are used to lay non-direction trails (Hölldobler & Wilson 1995), identify nest-mates (Detrain & Pasteels 1992), (Wilson 1976) or mark significant

objects (Turner 2011). Despite its ubiquity, chemical sensing is still a localised process, and ant is "...aware only of what reaches its antennae." (Gordon 1999). There is no "one-to-one correspondence between chemical and behaviour response." Indicating that chemical detection foes not control an agent directly and is simply a richer, more abstract sensor channel than the insects' poor eyesight. Howse (1970) makes the important point that chemical trail-laying methods entail "...no intentional communication of information, but that other termites are aroused and then quickly detect and follow the trail autonomously."

From this and similar comments (see Hölldobler & Wilson (1995) Sudd (1970), Gordon (1999), among others) we take chemical communications as a 'blind' mechanism. An agent might lay a trail or mark an object with the intending to influence other agents, but it does not necessarily wait for a reply. The chemical communication channel is asynchronous, it may or may not influence other agents at any time in its duration, while the original marking agent may be part of a group or wholly alone.

In this vein, we view chemical signals as analogous to vision in more complex animals – objects are identified as a class by chemical marker, rather than by shape or colour.

Other communications channels exist – Howse (1970) discusses a "tapping" system – but broadly, we view them all as similar concepts and ignore more direct agent-to-agent signals as uncommon and insignificant in comparison.

Locomotion and Manipulation "The three most important effector systems in ant behaviour are the mouth-parts, the legs and the sting" (Sudd 1970). In this work, we shall ignore the sting and focus on the other two.

An ant's mandibles are broadly applied tools and are used for other purposes than feeding: self-cleaning, holding prey, carrying nest-mates, larvae and eggs and excavation. In the same vein, an ant's legs have other uses besides walking, including digging (Sudd 1970). Termites have similar multi-function tools, with strong variance between casts, soldiers often sport "bizarre developments" (Gullan & Cranston 1994). These developments can range from the simply larger and more powerful, to the exotic, such as chemical spray nozzles and hole-plugging heads (Howse 1970).

Despite their broad utility, the mandibles are not mechanically complex. In ants, the mouth-parts take the form of a "set of cuticular limbs and flaps..." (Sudd 1970). In most species the mandibles are simply a pair of hinged jaws (Gullan & Cranston 1994).

Both species are equipped with six legs in a hexapod arrangement. "the centre of gravity of the moving insect always lies within [a] tripod, giving great stability...anchorage to the substrate needed to provide a lever to propel the body is through pointed claws and adhesive pads..." (Gullan & Cranston 1994). We consider this to imply that any powerful, stable locomotion system could roughly approximate insect movement.

For more detail on insect locomotion, see Chapman (1998).

In this work we shall ignore the caste differences – winged locomotion and the more exotic termite soldier castes are not considered. We like-wise overlook reproductive castes and conflate soldier (or ant "major") castes with workers (see Chapter 5 for a discussion on the future possibility of multi-caste swarms).

As for the workers, we approximate them with this summation from Sudd

Figure 2.1. Exposed plaster cast of the above ground tunnel network of a *Macrotermes michaelseni* mound. From Turner (2011), page 23.

(1970); the legs to be the source of the forces ants apply to their world, forces which are transmitted through the mandibles. We assume that a insect-like agent has only one set of manipulator and one set of locomotion effectors and that both are mechanically simple.

2.2 Selected Insect Swarms

The accomplishments of insect swarms are often massively larger than their constituent agents, both in physical size and complexity.

Construction The dwellings of eusocial insects are one of their most impressive accomplishments. In termites especially, the nest and associated outworks can grow monstrous.

Mounds on the order of several meters high are common for species of the genus *Macrotermes* (Turner 2011) (see Figure 2.1). This construction can house up to one or two million individuals (Howse 1970) and "is a device for capturing wind energy to power respiratory gas exchange for the colony," functioning much like a mammalian lung (Turner 2011).

While *Macrotermes* may be the most monolithic builders, Howse (1970) mentions some other impressive termite architectural achievements. *Procubitermes* and *Cubitermes* have been known to build nests with overhangs and caps for rain protection (while *Cubitermes* who live in drier regions do not). The European *Reticulitermes flavipes* produces a "carton-like material of wood, earth and faecal particles" which it uses to bridge substances that it cannot eat, allowing it to invade new food sources around obstacles. Finally, the Australian *Mastotermes darwiniensis* "in other ways [the] most primitive of termites" build nests of large horizontal cells, spreading over "almost 100 meters".

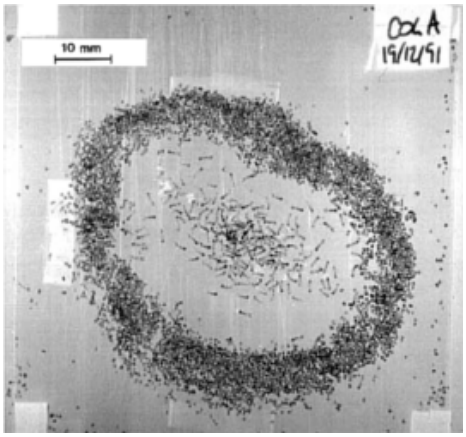

Figure 2.2. The circular nest of the *Leptothorax tuberointerruptus* ant in a laboratory environment. Note the dense cluster of ants in the centre and the entrance at 7 o'clock. From Franks & Deneubourg (1997) page 781.

There are many other examples, but in general it can be concluded that many species of termite take their nest construction seriously.

Smaller scale nests are produces by the the *Leptothorax* ant (see Figure 2.2). These much smaller swarms (up to 500 workers), produce a standard form of nest between two close fitting planes (see Figure 2.2. The Queen and brood occupying the centre with one entrance to the ring (Franks & Deneubourg 1997).

Both the massive termite nests and the smaller *Leptothorax* nests are constructed by broadly similar agents, cooperating as swarms.

Foraging On a similar massive scale to termite construction are the migrations and foraging of the army ant *Eciton bruchelli*. This ant builds no nest, instead the whole colony (which may consist of 150000 to 700000 workers) (Wilson 1979) crouches against a log or tree root at night and forms a living shield of ants "chains of bodies... accumulating layer upon layer" up to a metre across, within which the queen and the larvae are sheltered for the night (Wilson 1979).

During the day *Eciton bruchelli* is a 'swarm-raider', forming a broad swarm front that "brings disaster to practically all animal life in its path," the approach of which is loud enough to be audible (Hölldobler & Wilson 1989). This is illustrated in Figure 2.4, note the scale of the swarm front.

This daily migration cycle is periodically interrupted to allow the queen to lay eggs, the hatching of which galvanises the colony to continue its migrations (Hölldobler & Wilson 1995), (Wilson 1979).

Disaster Response Many species of both ants and termites are capable of aggressive military action.

The European ant *Pheidole pallidula* is able to respond with surprising flexibility to the incursions. The number of ants recruited for colony defence showed a marked relationship to the number of majors ('soldiers') in the intruding force,

Figure 2.3. A swarm of army ants forming a living bridge whilst on emigration. From Wilson (1979) page 62.

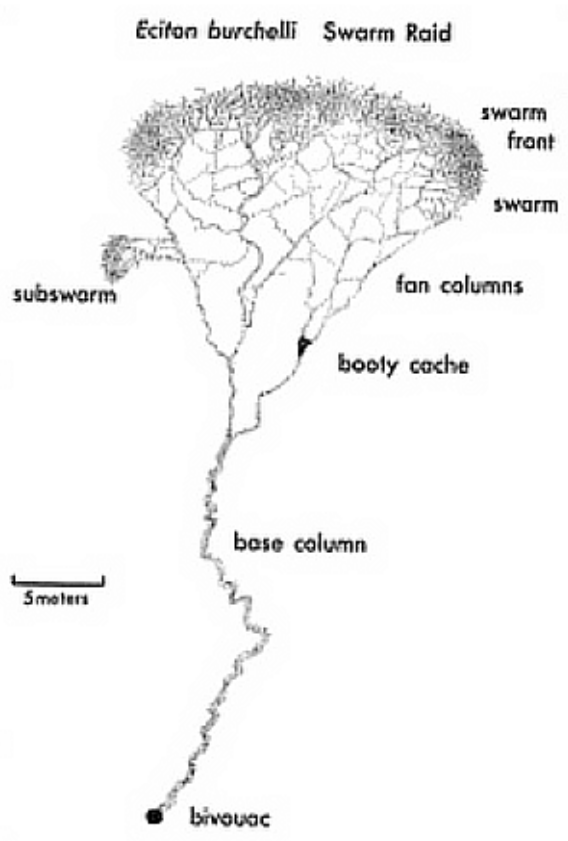

Figure 2.4. The raiding swarm of the army ant *Eciton burchelli*. From Wilson (1979) page 58.

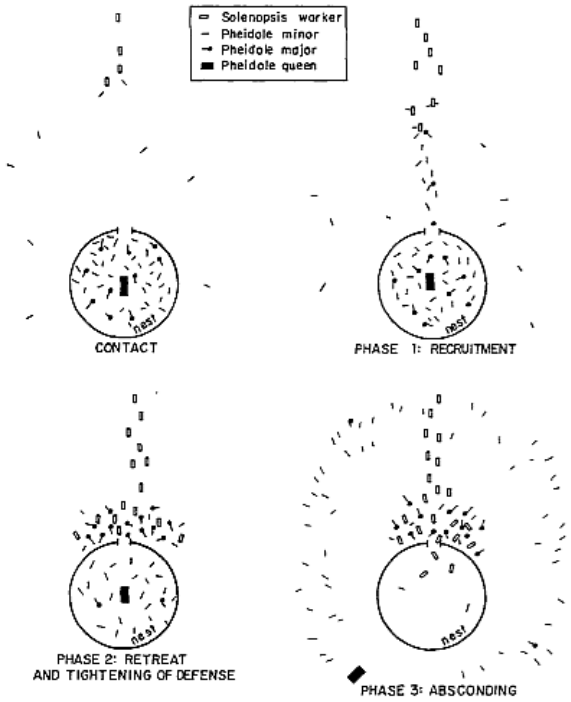

Figure 2.5. An abstract representation of the three phases of colony defence for the *Pheidole dentata* ant. Taken from Wilson (1976)

allowing the defenders to deploy their forces to best tactical advantage (Detrain & Pasteels 1992).

Once the defensive force was mobilised, the ants showed a "... very efficient defensive strategy of cooperative attack: minors immobilised the intruders by seizing their legs... whereas majors' attacks were concentrated against spread-eagled enemies, killing them with their powerful mandibles" (Detrain & Pasteels 1992).

A similarly sophisticated tactical doctrine is followed by the *Pheidole dentata* ant in defence of their nest against invading fire ants. Hölldobler & Wilson (1995) describes a strategy in three phases, the active phase dependant on the numbers of invaders, outcome of the battle and the enemy's deployment. These three phases are illustrated in Figure 2.5, see also Wilson (1976).

The collective enterprises undertaken by the colony are all undertaken by large numbers of individuals. Each task is complex, with interconnecting components that must be coordinated and completed in parallel all in a dynamic environment, by *very simple individuals.*

This all being the case, it is nevertheless true that all these tasks, involving large numbers of simple agents, are *undertaken without any centralised controller and in the absence of a global communications net or planning system.*

We will now explore how this can be possible.

2.3 Stigmergy and Self-Organisation

In all eusocial insect swarms, the impression of centralisation and tightly coupled cooperation is an illusion. The great disparity of scale between insect colonies and human observers often leads to the assumption that such creatures must be centrally controlled by some dictator or conductor or are far more human-like than is the case (see Howse (1970) for examples of this anthropomorphism).

Instead, all cooperation is generated from local interactions. Every insect knows only what it can see and receives no explicit direction from the "Queen" – indeed, in an established termite nest the sole occupation of the "primary reproductive" is an egg-laying machine (McGavin 2001).

The coordinating information is implicit in the environment and actions of other agents. This is all that is needed to produce a society far larger than any one of its inhabitants could construct or even conceive on their own. "... Cognition is the emergent result of the collective dynamics of either interacting autonomous agents or basic control units in a single agent... the entities have no reference to the global pattern or cognitive phenomenon they are contributing to create"(Trianni, Tuci, Passino & Marshall 2011).

The governing principle of this coordination is *stigmergy*.

Originally introduced by Grassé (1959)[1], stigmergy is generally "used to indicate indirect communication mediated by modifications of the environment ..." (Dorigo et al. 2000).

More specifically:

> "... stigmergy is communication via long term traces, physical practical outcomes, useful environment modifications... Stigmergy is... where the addressee does not perceive the behaviour (during its performance) but perceives post-hoc traces and outcomes of it. In order for a trace-based communication [to] be stigmergy it is necessary that the perceived "object" be a practical one and the originating action also be for practical purposes (like nest building)." (Castelfranchi 2006)

Stigmergy is a blind, asynchronous process. Explicit communication of information from agent-to-agent does not occur. Instead, agents are required to respond to stimulus as they perceive it. As Dugatkin & Reeve (1998) observe (similarly, see Deag (1980)):

> "communication is said to occur when acts or cues given by one individual influence the behaviour of another."

Stigmergy is the principle driving component behind the "self-organisation" of the swarm. The "set of dynamic mechanisms whereby structures appear at the global level of a system from interactions among its lower-level components, without being explicitly coded at the individual level." (Garnier et al. 2007).

Thus "stigmergy is essentially a mechanism that allows an environment to structure itself through the activities of agents within the environment: the

[1] The author cannot actually read French and a translation of this paper is surprisingly hard to find. It is included in the citations for the sake of completeness, the actual description of stigmergy used in this work is drawn from other, later interpretations. Chiefly Dorigo, Bonabeau & Theraulaz (2000)

state of the environment and the distribution of agents within it, determine how the environment and the distribution of agents will change in the future." (Holland & Melhuish 1999).

Grassé (1959) showed that coordination and regulation of termite building activities follow this paradigm. Dorigo et al. (2000) discuss this and other examples in some detail (see also Turner (2011) for termite building).

These examples rely heavily on the insects laying and detecting pheromones. Chemical markers identify objects, the presence of which may excite behaviours. The natural decay of the volatile markers provide the negative feedback necessary to ensure stability (Dorigo et al. 2000), (Garnier et al. 2007).

These markers are deliberately placed by agents in the course of task execution, implying an explicit intent to communicate. However, chemical markers could also be viewed as allowing very limited agents to identify objects in context rapidly, without the need for a broad information base (as discussed above, see also Howse (1970)). Taking this view, stigmergic reaction to "perceived practical objects" (Castelfranchi 2006) takes place at a slight abstraction, where objects are already identified according to class. The author believes that this is implicit in the examples in Dorigo et al. (2000), Garnier et al. (2007) and Turner (2011) and that chemical ques replace object and context identifiers that humans would recognise visually.

Critically, chemical signals do not have inherent bandwidth limitations. As they are dependant on other agents to observe them, any number of insects can "broadcast" a signal at once.

Such stigmergic systems are also termed "self-organising". Garnier et al. (2007) identify four basic ingredients in self-organisation (see also Dorigo et al. (2000):

1. Positive feedback that results from execution of simple behavioural "rules of thumb" – the actions of one agent encourage other agents, encouraging more agents, and so on.

2. Negative feedback counterbalancing the positive, leading to system stabilisation, preventing all agents from being monopolised in the one task. This is generally due to reduction in stimulus.

3. Amplification in fluctuations by positive feed-backs. Random fluctuations thus become the seeds from which structures nucleate and grow.

4. Direct stigmergy interactions among individuals "to produce apparently deterministic outcomes".

We can use the ideas of self-organisation and stigmergy to explain all the insect cooperation examples discussed previously (Section 2.2):

- *Nest Construction* – Turner (2011) and Dorigo et al. (2000) describe the "bucket-brigade" process used by termites to collect dirt-pellets and stack them in pillars. Soil pellets are impregnated with a pheromone to identify them as an object type. Individuals then deposit these randomly. Coordination commences once deposits of pheromone tagged pellets reach a critical size, once this occurs the probability of other agents adding to this pile increases, which in turn creates a larger pile and greater stimulus for the agents (see Figure 2.6).

Figure 2.6. Termite employing stigmergic building to construct pillars. From Turner (2011) page 27.

In this process, the presence of pellet mounds is the stigmergic trigger (for an agent to drop its payload), randomly moving agents that do not find a pellet mound provides the negative feedback (Dorigo et al. 2000) and the increasing size of the mounds (correspondingly increasing the chances of a randomly moving agent finding the mound) provides the positive feedback. A single pair of pellets is the initial fluctuation needed to start this feedback.

The circular plate nests of the *Leptothorax* discussed above are built in the same decentralised fashion. Franks & Deneubourg (1997) describe how the "cluster of adult workers around the carefully sorted brood cluster serves as a mechanical template to determine where the nest wall should be built... the first building workers who make contact with the cluster... 'pace out' a relatively short distance before depositing their building material..." Although they stress that the term 'template' "... does not imply that the template provides individual workers with global knowledge... for the individual worker... the template can be thought of as providing a locally restricted boundary zone where building is most likely to occur."

Franks & Deneubourg (1997) describe how small numbers of workers collect building material from the environment, return to the nest and release their prize once they have made contact with a cluster of nest-mates or previously deposited stones. If the latter, "the ants seem actively to use the stone they are carrying as a battering ram and release their stone only after they have felt the resistance of other stationary stones." Ants that remain in the nest are "frequently seen to pick up stones that are close to them and bulldoze them outwards again." The increasing density of stones provides the negative feedback required to produce the coherent shape.

In both cases, building is an individual effort. Single agents make use of local observations – the presence or absence of objects and their physical locations – to move tiny components into place. The nest is the sum of these activities.

- *Foraging* – Trail-following is inevitably the result of auto-catalytic mech-

anisms (Deneubourg & Goss 1989). Even the very large-scale *E. burchelli* swarms rely on the interpretation of markers and events by individuals.

Once the increasing light of dawn has advanced past a specific threshold, the bivouac disperse and workers run onto new ground, laying down a chemical trail as they do so (Wilson 1979). This trail prompts more workers, repeating the "snowball" effect. Once the column has formed, there is no leader individual. Instead, "workers finding themselves in the van press forward for a few centimetres, then wheel back into the throng behind them, to be supplemented immediately by others who extend the march a little farther." (Wilson 1979).

- *Dissaster Response* – The ant's defence of the nest is accomplished through chemical trails and ques, (Detrain & Pasteels 1992), (Hölldobler & Wilson 1995), (Wilson 1976).

  Pheidole pallidula recruitment is governed by trail-laying behaviour of the part of *Pheidole pallidula* minors. The defensive strategy "can be generated by the elementary behaviour of recruiters" (Detrain & Pasteels 1992) The more invaders present, the more *Pheidole pallidula* are likely to encounter them, creating greater numbers of recruiters who lay more and (by repetition) stronger trails. The emission by one ant of "alarm" signals stimulates the recruitment of other defenders, who in turn also signal alarm, resulting in another snowball effect.

2.4 Threshold Models

While the self-organising characteristics of insects have obvious use in building swarms, a broader unifying structure is required to apply stigmergic cooperation to a general robotic architecture. This is provided by the concept of behaviour thresholds. Simply: an insect is controlled by a finite number of behaviours (on order of 40 to 100 according to Hölldobler & Wilson (1995)). The activation of any one of these behaviours is dependant on external stimuli (such as the existence of objects, chemical signals or other sensory stimulus) and internal thresholds. Different castes and individuals may have different thresholds for the same stimulus-reaction pair.

As Theraulaz, Bonabeau & Denuebourg (1998) remark:

> "Colony-level flexibility in response to external challenges and internal perturbations is an essential feature of division of labour in social insects. Simple response threshold models, which assume that each worker responds to a given stimulus when stimulus intensity exceeds the worker's threshold, can explain how flexibility at the colony level results from the workers' behavioural flexibility"

However, they emphasise that these thresholds are not fixed, as a rigid threshold implies "that individuals are differentiated and roles preassigned". Furthermore, it cannot account for "robust task specialisation within physical or temporal castes" and is inconsistent with observations. Consequently, they develop a stimulus threshold model with a dynamic threshold (the following is adapted from Theraulaz et al. (1998)):

Assume that there are N workers, each denoted by i. They have response threshold $\theta_{i,j}$, where j is a single task of the tasks i may attempt. The stimulus for i to engage in task j is given by $s_{i,j}$.

An individual i will engage in task j with a probability:

$$T_{\theta_{i,j}} = \frac{s_j^2}{s_j^2 + \theta_{i,j}^2} \tag{2.1}$$

Theraulaz et al. (1998) appear to assume that the stimulus to do task j is universally recognisable for all agents.

Let ξ and φ be coefficients that describe the "learning" and "forgetting" rates of the threshold (respectively), or the rates at which the threshold decreases through execution of j or increases as j is attempted.

Thus if agent i is performing task j over Δt, the threshold becomes:

$$\theta_{i,j} \to \theta_{i,j} - \xi \Delta t \tag{2.2}$$

While if agent i is not executing task j over Δt, the threshold becomes:

$$\theta_{i,j} \to \theta_{i,j} + \varphi \Delta t \tag{2.3}$$

If $x_{i,j}$ is the fraction of time spent by agent i doing task j, then within Δt agent i does task j for $x_{i,j}\Delta t$ and other tasks for $(1 - x_{i,j})\Delta t$.

Therefore, the resultant change in $\theta_{i,j}$ within Δt is:

$$\theta_{i,j} \to \theta_{i,j} - x_{i,j}\xi \Delta t + (1 - x_{i,j})\varphi \Delta t \tag{2.4}$$

Theraulaz et al. (1998) assume that ξ and φ are the same for all insects and tasks. They also bound $\theta_{i,j}$ between θ_{min} and θ_{max}.

For the development of a general robotic architecture Equation 2.1 is the most significant. This equation provides a mechanism for behaviour selection, providing a decision making algorithm that is responsive to stimulus (s) and incorporates past individual experience in the form of a moving threshold (θ). We will not investigate the details of Theraulaz et al.'s (1998) implementation much further save to note some important points:

- An active individual gives up on a task with the probability p every time step. Thus the average time an individual spends of task j is $\frac{1}{p}$. Although it may re-engage in the task immediately if "stimulus intensity is still large".

  In their implementation Theraulaz et al. (1998) fix this value as constant for all tasks and individuals.

- The "average temporal dynamics" of $x_{i,j}$ includes a Gaussian stochastic term that "...simulates the fact that individuals encounter slightly different local conditions."

- They also assume "...for simplicity that the demand for each task increases at a fixed rate per unit time..."

Mathematical descriptions of the change in $\theta_{i,j}, x_{i,j}$ and s_j can be found in Theraulaz et al. (1998), Equations 4, 5 and 7.

The results of their model are interesting. Broadly speaking, they illustrate that their variable threshold model can produce specialisation among individuals, with the thresholds allowing for "learning in the form of a reinforcement process... this extended model could account for the genesis of task allocation and within-caste specialisation." They also conduct an examination as to how "learning and task-switching rates are expected to affect specialisation." They illustrate that individuals with a low θ_j have a correspondingly high x_j value (and visa-versa). Also, if an individual has an initially low θ_j value, they are more likely to become a specialist in that task. Finally, they demonstrate that if specialists ("individuals with a low $\theta_{i,j}$") are removed, "individuals with previously high $\theta_{i,j}$ lower their $\theta_{i,j}$ and become more responsive to task associated stimuli" showing that the model incorporates some of the inherent flexibility of the swarm.

We are not especially interested in how this model integrates with biological swarms in detail. For the purposes of this project, the threshold model becomes a fundamental idea around which we shall construct a software architecture and it provides a starting place from which the design flows. As we shall see, ultimately it has little effect on the execution of the simple tasks upon which the design is tested. However, its importance should not be underrated. The threshold model represents the most concise, general and mathematical description of swarm decision making encountered in this reading.

For more detail on the threshold model and examples of implementation see: Schmickl & Crailsheim (2008*a*), Garnier et al. (2007) and Dorigo et al. (2000). For a more complex investigation incorporating other factors (such as insect age) see Merkle & Middendorf (2004).

2.5 Swarm Robots, Tasks and Control

The above points have obvious correlation to the eusocial insects discussed previously. However, many swarm robot projects seek to implement reduced swarm agents that focus on one task or aspect of cooperation.

We shall review several examples, illustrating how aspects of the swarm cooperation of eusocial insects may be realised in hardware. We begin with the simplest agents and tasks, moving upwards in task and hardware sophistication. All of these tasks and robot configurations are of interest in the development of a general swarm architecture.

2.5.1 Simple Swarms and Aggregation

First, we will examine basic agent grouping and clustering tasks. These bring the agents together physically, encourage the spread of information and demonstrate collective decision making (Garnier et al. 2005) and represent real-world swarm agent hardware at its most basic.

Garnier et al. (2005) created a relatively large swarm[2] comprised of the extremely small and relatively primitive "Alice" robot. The "Alice" was essentially

[2]Up to twenty individuals, which is a fair number of actual robots. The most spectacularly large physical swarm is the 1024 "Kilobots" of Rubenstein, Cornejo & Nagpal (2014), but this very much the exception. Due to time, cost and other constraints real-world laboratory swarms tend to be fairly small in absolute numbers and tiny compared to insect hives.

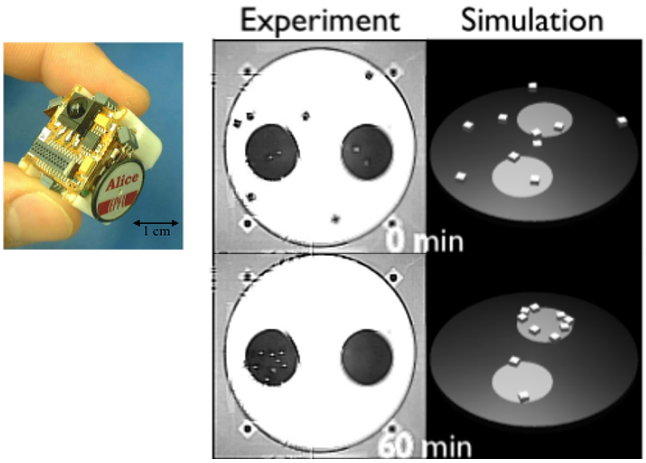

Figure 2.7. Top left: The Alice robot used in the cockroach aggregation experiments of Garnier et al. (2005), From Caprari & Siegwart (2005) page 1. Right: the series of stills from the aggregation experiments. The left-hand sequence have show the Alice robots, they appear as black squares on the white arena and faint black squares with white dots in the black aggregation shelters. Notice how all the robots are within the left-hand shelter by the end of the sequence. From Garnier et al. (2005) page 4.

a pair of wheels with four infrared sensors for local avoidance communication. The sensors had a maximum range of four centimetres. The whole robot was controlled by the diminutive PIC16LF877 8-bit micro-controller (see Figure 2.7).

The aim of the project was to replicate a behavioural model for the aggregation of the *Blattella germanica* cockroach, which clusters together in shelters. This use of real or simulated robots to approximate aspects of insect behaviour is common[3]. The behavioural model used by to achieve this was also highly simple. Groups of ten or twenty individuals were deployed in a white arena fifty centimetres across (see Figure 2.7). From analysis of live cockroaches, it was known that when placed in an open arena, the insects perform a "random walk" (see Fujisawa, Dobata, Sugawara & Matsuno (2014), Holland & Melhuish (1999) Labella et al. (2006), Ishiwata et al. (2011) for other projects employing the random walk as the basis for agent interaction).

Once the Alice cockroaches reached the edge of the arena, they followed the wall, with a constant rate to return to the centre of the arena. The robots periodically stopped and waited, the rate at which they did this increased with the number of other robots they could detect (ID numbers were transmitted via the infrared sensors to allow robots to detect swarm-mates).

This simple system produced the self-organising positive feedback previously discussed; the more robots in a given area, the more likely for another agent

[3]For example: Phan & Russell (2012), Payton, Estkowski & Howard (2005), Ishiwata, Noman & Iba (2011), Cicirello & Smith (2001), Duarte, Christensen & Oliveira (2011), Kallel, Chatty & Alimi (2008), Wagner, Altshuler, Yanovski & Bruckstein (2008), Lee, Chong & Defago (2007) and Schmickl et al. (2007) and Schmickl & Crailsheim (2008*b*) among many others.

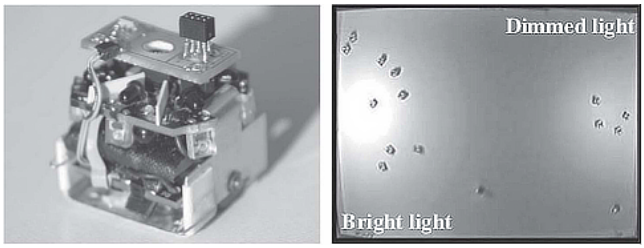

Figure 2.8. Left: the "Jasmine" robots used in Bodi et al.'s (2012) bee clustering experiment. Right: the robot "bee" swarm, aggregating around the light source. From Bodi et al. (2012) page 90.

to encounter them and stop, increasing the group. In this project there was no counter-balancing negative feedback.

Garnier et al. (2005) expanded their experiment by modifying the robots' controllers so the robots only halted when the ambient light fell below a certain threshold – to emulate cockroach shelter finding behaviour. They found that "...the group of robots will choose preferentially a shelter that is sufficiently large to house all its members. But when the group is confronted with two sufficiently large shelters, the self-enhanced aggregation mechanism can lead the group to two stable choices, with a preference for the larger shelter." The randomly moving robots are more likely to encounter the larger shelter. It is important to note that making an objective judgement, or quorum decision on which shelter to favour, is beyond the abilities of the swarm agents. The decision is implicit in the the interactions of the robots.

A similar clustering experiment was conducted by Bodi et al. (2012) employing the "Jasmine" micro-robot[4] to model bee clustering behaviour. It resulted in comparable outcomes.

Both these robots are incomplete swarm agents, they can have perception and locomotion, but do not manipulate objects. However, their physical simplicity demonstrates that these aspects of swarm agents do not need to be highly sophisticated. They also demonstrate the use of the agent as a stigmergic signal.

Other minimalist swarm robots are also employed in the work of Rubenstein, Cabrera, Werfel, Habibi, McLurkin & Nagpal (2013), Adouane & Le Fort-Piat (2004), Kernbach (2013) and Schmickl et al. (2007), among others.

2.5.2 Manipulative Swarms

Parker & Zhang's (2006) swarm of bulldozing robots are perhaps the first step between simple grouping robots and manipulative swarms. Designed to clear a two-dimensional area by explicitly copying the nest building behaviour of the *Leptothorax* ant (described by Franks & Deneubourg (1997) and discussed previously, see Section 2.2)[5]).

[4] See Figure 2.8 or `www.swarmrobot.org`.

[5] Although Parker & Zhang (2006) refers to *Leptothorax albipennis* and Franks & Deneubourg (1997) speak of the *Leptothorax tuberointerruptus*. The author is unsure if they are different creatures, but as both describe the same behaviour, for our purposes this is not a significant distinction.

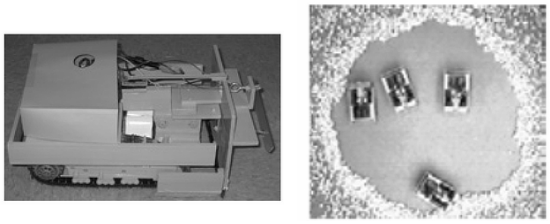

Figure 2.9. Left: the robotic bulldozers of Parker & Zhang (2006), built from a child's toy. Right: the cleared "nest". From Parker & Zhang (2006) pages 14 and 16 respectively

These agents were among the simplest physical robotic agents encountered in this research. Stripped-down toy bulldozers, each equipped with bumper switches, a force sensitive plough and a micro-controller, they clearly demonstrate the power of limited agents; over the course of two hours four of these machines, starting in a clear zone 60 centimetres in radius, were able to sweep clear an area of floor 90 centimetres in radius (see Figure 2.9).

To do this Parker & Zhang (2006) simply rely on the force sensitive plough with which their agents shovel their "nest" building material (landscaping gravel). As the robots wander out of the initial clear centre of the arena, they push gravel in front of them. As this continues, the gravel in front of any one robot becomes more compact, exerting more pressure on the plough. Eventually, this pressure exceeds a threshold (as registered by the plough force sensor) and the robot considers ploughing to be "finished" in this direction. It turns and selects a new direction. Robots explore their environment by simply moving in a straight line, turning and avoiding other robots.

This produced a robot with three states "finishing", "ploughing" and "colliding". Because there are only two classes of object in the environment – gravel and other robots – the extremely simple sensors are sufficient to determine the correct state change. However, this represents a step up from the bees and cockroaches of Garnier et al. (2005) and Bodi et al. (2012) in that Parker & Zhang's (2006) machines can identify a non-robot object and manipulate it, sending a stigmergic signal to the other robots at the same time.

The robots thus use stigmergy to make decisions and cooperate; if one robot has "finished" the nest in a given direction the resultant compacted gravel will be the signal to other robots that no further work in that direction can be undertaken. These are the "...useful environment modifications..." that convey information to the other members of the swarm (Castelfranchi 2006).

However, the agents are fully autonomous; as Parker & Zhang's (2006) results show, one machine can complete the nest building task as well as a group, just in a longer time.

Ijspeert et al.'s (2001) "stick-pulling experiment" takes manipulation further. In this experiment[6] a circular arena, 80 centimetres in diameter and possessing four holes in a square formation, was populated with groups of two

[6]Martinoli et al. (2004) also discuss this project, but under a different, earlier citation. The experiment is virtually identical and the implication is that Ijspeert et al. (2001) is follow up work.

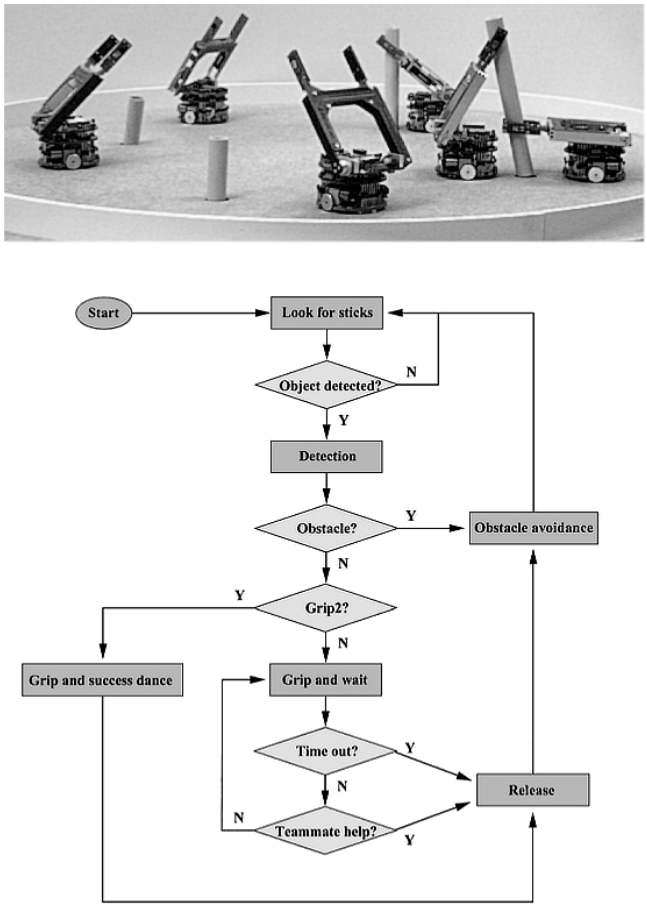

Figure 2.10. The stick-pulling experiment. Top: the robots in action, bottom: the hard-coded controller in each robot. Note the two robots on the top-right cooperating to remove the stick from the table hole. Bottom: the robot controller's flow diagram. From Ijspeert et al. (2001) pages 152 and 153 respectively.

to six "Khepera" robots[7], each equipped with a gripper and infrared proximity sensors (Ijspeert et al. 2001). Objects are identified by their "width" in these sensors; robots are wider than the target sticks.

The robots' task was to remove four fifteen centimetre long sticks from the corresponding holes in the arena. Each hole contained one stick, projecting five centimetres from the table-top when at rest. The relative dimensions of the robots' grippers and the sticks ensure that a single agent cannot free a stick by itself and depends on the cooperation of other robots to complete its task.

At no time do the robots communicate explicitly, nor do they recognise each other as individuals. Instead they rely upon the changes caused by the other robots, for example: a stick in the grip of another robot requires more force to move than a free stick. The sensors are limited, their view is poor (a stick held by another robot can only be recognised as such when approached from "the opposite side, within a certain angle...") and manipulation simple. Their controllers are hard-coded (see Figure 2.10).

This system illustrates stigmergy in a more complex sense than Bodi et al.'s (2012) bees or Garnier et al.'s (2008) 'roaches. The actions of another robot cause a target object to change its state, to be come a captured stick, rather than a free stick (see also Labella et al. (2006), Ishiwata et al. (2011) and Liu et al. (2007) for other controllers reliant on such change in object state).

Although each machine is autonomous, a single machine would be fully capable of meandering around the arena, grasping sticks and letting the grip time parameter expire on its own. It is the interactions between pairs of these autonomous machines that provide the information necessary to complete the task. Martinoli et al. (2004) refers to this stigmergic coordination as communication via a "shared blackboard", which is the environment. They also emphasise that this project is an instance of "swarm intelligence" referring to the "biological examples provided by social insects."

The stick-pulling robots also exemplify a key swarm concept: temporal order is implicit. The robots with the sticks have no means of calling other agents to help them, they must wait until their need is recognised. If a swarm task has a critical execution order, then it is assumed that there are enough agents that one will eventually identify the need and perform the next step. The task 'hangs' until this occurs.

Thus there is a strong relationship between agent timeout, swarm size and cooperation rate (Ijspeert et al. 2001), demonstrating the need to appreciate intuitive probabilities in swarm cooperation. The probability of agents encountering each-other and interpreting the current environmental cues must be sufficiently high.

At a similar level of complexity to the stick-pulling project is the leaf-curling project of Phan & Russell (2012). In a successful attempt to replicate the activities of the weaver ant (see Figure 2.11), this project again demonstrates simple manipulative agents and communication "via long term traces, physical practical outcomes, useful environment modifications..." (Castelfranchi 2006).

Once again, the agent is a simple wheeled machine with "infrared-based sensors" and a single gripper. Neither were Phan & Russell's (2012) robots "equipped with any wireless, infrared or similar communication devices... the

[7] see Figure 2.10 or `www.k-team.com`, although the exact robots used in this particular experiment appear to be discontinued.

Figure 2.11. Phan & Russell's (2012) leaf curling robots (right) and the weaver ants they were modelled upon. From Phan & Russell (2012) page 103

simplicity of [the robots] is also emphasised by not allocating memory to remember the robot's path nor [by] providing any ability to recognise specific robots." Again the task is beyond the capabilities of any one robot.

See also Holland & Melhuish (1999) Ishiwata et al. (2011) for further examples of insect-like swarm robots and tasks.

Both the "stick-pulling" and "leaf-curling" projects demonstrate swarms where the agents are strongly dependant on each-other. The half-complete task is in itself the communication channel. Other examples of this stigmergy through construction approach can be found in Werfel et al. (2006) Matthey, Berman & Kumar (2009). Both these projects use local interactions among construction materials and agents to produce finished, two-dimensional structures. However, they also use active communication between the agents and the building components. Both projects employ magnets for collection and carefully defined construction blocks in assembly.

If the environment itself is the only communication channel, perception becomes critical. The robots must be able to interpret the environment accurately; classification of objects and object states must be real-time and unambiguous. In most the projects examined so far this is facilitated via essentially empty environments. The stick-pulling and leaf-curling robots, as well as the building robots of Werfel et al. (2006) and the foraging robots of Labella et al. (2006) exist in an environment comprised of target objects, obstacles (arena walls) and other robots. Limited sensors can thus be employed without much danger.

2.5.3 Recruitment, Signalling and Advanced Swarms

The next phase in complexity are swarms that permit robots to recruit other agents, to manipulate objects collectively and, frequently, to transmit information through the swarm by simple signalling schemes. Ultimately, these advanced cooperatives lead out of the swarm field altogether, becoming cooperating groups of complex robots, rather than true swarms.

The foraging task of Nouyan et al. (2009) Typifies many of these advanced swarm features. This project creates a swarm of several *s-bots* (see Figure 2.12) "that can operate both autonomously and as a group"[8].

Foraging is a common swarm task (see Liu et al. (2007) and Labella et al.

[8]See also the related work of Mondada, Gambardella, Floreano, Nolfi, Deneuborg & Dorigo (2005).

(2006)), but both Mondada et al. (2005) and Nouyan et al. (2009) emphasise the ability of the individual *s-bots* to form physical links with other robots in the swarm and the importance of these connections in solving physical tasks.

As they do so, they achieve "teamwork" and demonstrating a "division of labour characteristic of social insects" (Nouyan et al. 2009). Their chains carry echos of the army ant chains discussed in Wilson (1979).

In Nouyan et al.'s (2009) work, a swarm of up to twelve autonomous robots is used in a foraging and recovery task, locating a "prey" object and returning it to a "nest". This is done under the constraints of swarms:

1. The prey requires concurrent, physical handling by multiple robots – it is too large and heavy to be moved by one robot alone.

2. The robots have a small perceptual range compared to the distance between the nest and the prey. Perception is also unreliable.

3. No robot has any (explicit) knowledge about the environment beyond its perceptual range.

4. Communication among robots is unreliable and limited to a small set of simple, local signals.

Nouyan et al. (2009) rely on the robots cooperating as a much closer unit than customary. A sequence of stills from this process is depicted in Figure 2.12. The *s-bot* robots explore their environment and form "chains" between the nest and the prey. These chains form paths along which other *s-bots* transport the captured prey to the nest – in a process reminiscent of the chemical trail markers of ants and termites, although a physical robot is needed to form each link in the trail, with the robots themselves becoming "landmarks or beacons" (Nouyan et al. 2009).

A foraging task requiring looser cooperation may be seen in Labella et al. (2006).

The *s-bots* are larger and more complex than most swarm robots, but they do not depart from the sense, move and manipulate paradigm. Although the *s-bot* had "five degrees of freedom (DoF)..." two DoF for the wheel/treads system, one to rotate the *s-bots* upper part "called the turret", one for the grasping mechanism and a final DoF for elevating the grasping mechanism (Nouyan et al. 2009).

Although clearly possessing more mechanical complexity than the previous examples of swarm agents, the *s-bot* is really an elaboration on the same idea: the locomotion system, although unorthodox, is still a differential drive system as previously, just more robust. The other degrees of freedom are clearly connected with the more complex gripper, which is more general than the system used by Phan & Russell (2012) and more robust than that of Ijspeert et al. (2001).

The *s-bot* really sets itself apart with a much more complex sensor suite. Nouyan et al. (2009) discusses a variety of proximity sensors, providing readings around the turret and underneath the robot[9]. Manipulator sensors, on both the gripper, the elevation mechanism and the turret, provide more sophisticated

[9]Nouyan et al. (2009) has fifteen and four of these (respectively), while Baldassarre et al. (2007) mentions sixteen and four.

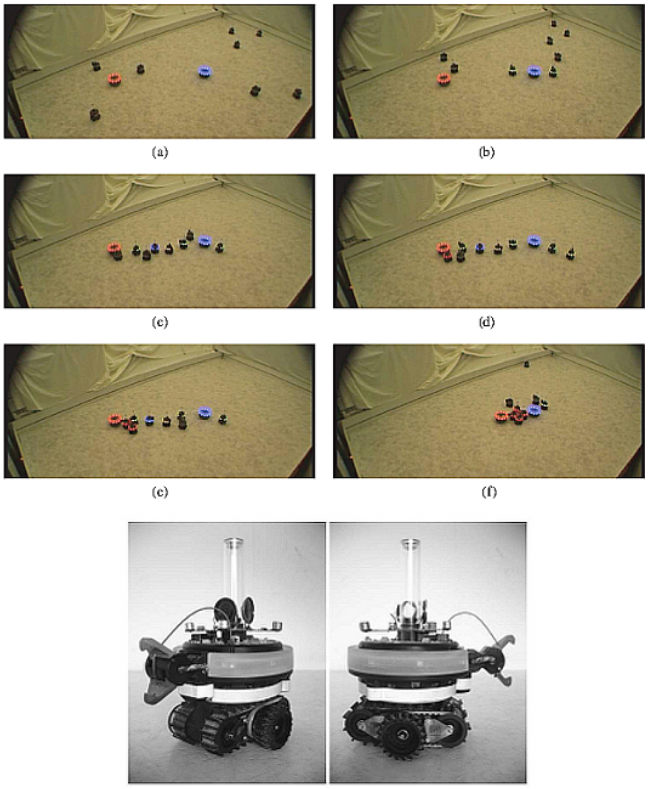

Figure 2.12. Top: stills from the foraging task of Nouyan et al. (2009), showing eight *s-bots* moving the red "prey" to the blue "nest". From Nouyan et al. (2009) page 703. Bottom: the *s-bot*. From Baldassarre et al. (2007) page 255.

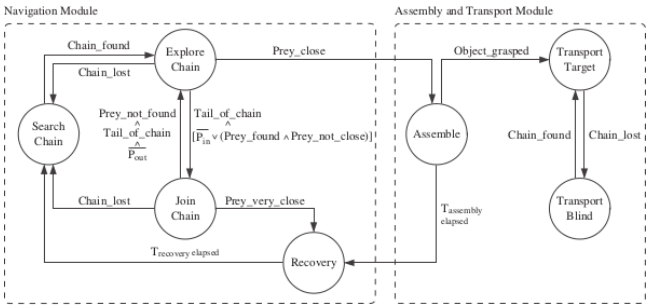

Figure 2.13. The controllers for the *s-bots* of Nouyan et al. (2009). The three modules of the foraging task are boxed, with circles representing the states (behaviours) of a finite state machine. The labelled arrows signify the state change triggers. From Nouyan et al. (2009) page 700.

control over manipulation. Robot orientation is determined by a three-axis inclinometer and the local environment is perceived by a unidirectional camera and eight light sensors. Unusually, the robot is also fitted with four unidirectional microphones and a pair of speakers. More conventional local communication is provided by eight RGB LEDs distributed around the robot. Finally internal motor information "such as torque, position and speed" is also available. All of this combines to give the robot a much richer perception of itself and its environment, illustrating that while swarm agents should be mechanically simple, they need not be limited in their sensors or software.

Nouyan et al.'s (2009) *s-bots* depart from simple projects with their signalling abilities. Neither Ijspeert et al.'s (2001) nor Phan & Russell's (2012) machines could signal with more than their physical presence – a change in the environment, rather than a change in any properties of the agents. In contrast Nouyan et al.'s (2009) project requires that agents change their physical properties to identify them as a class. Thus they can identify the agents who comprise the trail from the "prey" to the "nest". This reflects the increasing complexity of the project – there are more possible states in the environment, the simple presence or absence of objects does not necessarily convey sufficient information for coherent cooperation. By using their LEDs to send broadcast signals, the *s-bots* employ a local signalling system akin to the chemical pheromones of real insects.

Other projects that employ local signalling in this or a related fashion are Schmickl et al. (2007), Payton et al. (2005).

These projects advance upon Nouyan et al. (2009) and create a swarm-wide communications net, created by robot-to-robot interactions. This approach is a common theme and is often termed "virtual pheromones" – although Schmickl et al. (2007) object that "... as biologists we do not think [this is] appropriate." They are probably correct, but the name has stuck.

The broad purpose of virtual pheromones in both these projects is to disseminate information throughout the swarm, to allow individuals to take advantage of their nest-mates' discoveries. This effectively represents an attempt to broaden the perspective of each robot. Like real insects, the virtual pheromones do not carry direct instructions, each robot is required to make decisions based upon the information available.

In simpler swarms, such as those of Parker & Zhang (2006) and Ijspeert et al. (2001) information does not propagate directly, rather robots use a more limited form of stigmergy by interpreting objects in the environment.

Other projects employ similar semi-implicit, decentralised communications. One of the most notable is the "belief sharing" system of Isik, Stulp, Mayer & Utz (2007) (see also Stulp, Isik & Beetz (2006) and Buck, Schmitt & Beetz (2002)), where a robot's internal "belief state" is broadcast locally.

Despite the differences in hardware and perception, all the swarm robots examined so far have some common elements. They can recognise other agents – although not usually as individuals, but as members of the object type "other agents". All agents can detect obstacles – such as the walls of their arena – and they can usually detect at least one class of target "object of interest", the manipulation of which is essential to their task.

2.6 General Cooperative Architectures

Many previous projects have attempted to broaden robot cooperation, allowing the robots in the group to attempt several tasks without fundamental reprogramming. These projects usually produce a behaviour-based system and a central program infrastructure for selecting behaviours. In this they have much in common with insects. However, they are often aimed at non-swarm robots or tasks.

2.6.1 Robot Behaviours

> "A behaviour, simply put, is a reaction to a stimulus." (Arkin 1998)
>
> "... behaviours [are motor] control laws that encapsulate sets of constraints so as to achieve particular goals." (Matarić 1994)

Behaviours are fundamentally *reactive*, responding to what occurs rather than relying on a master plan. "Causality flows into the system from the world, drives the roles which choose what to do, resulting in action which changes the world and back again into the system, which responds to the changes... An agent engaging in a routine [behaviour] is not driven by a preconceived notion of what will happen. When circumstances change, other responses become applicable..." (Agre & Chapman 1987).

Robotic behaviours are often traced back to the "Subsumption" architecture of Brooks (1999) and are usually the basis of general cooperative architectures. Brooks advocated for a horizontal approach to robot programming, where a series of simple behaviours are implemented in parallel, rather than attempting to abstract and deliberate on the robots' environment.

In Brooks' system, each behaviour was a simple Finite State Machine, with limited objectives. Numerous behaviours were implemented in layers, producing complex behaviours in the summation of their outputs. "Each module is an instance of a fairly simple computational machine. Higher level layers can *subsume* (italics added) the roles of the lower levels by suppressing their outputs. However, lower levels continue to function as higher levels are added." (Brooks 1999)

This is a priority-based arbitration system (Pirjanian, Huntsberger, Trebi-Ollennu, Aghazarian, Das, Joshi & Schenker 2000) and forms a linkage between action and perception, with cognition not being explicitly included in the algorithm and merely in the eye of the external observer (Brooks 1999). This has obvious parallels with the complexity of the insect hive – the hive's responses appear to be coordinated on a global scale, but this is an illusion created by the interaction of large numbers of autonomous agents. The global coordination of the hive's response is only apparent to the observer, while the finite-state behavioural structure also resonates with the threshold model of Theraulaz et al. (1998).

Brooks originally outlined nine dogmatic ideas upon which the design was based, which were subsequently refined and are summarised in Arkin (1998) as follows (slightly paraphrased):

- Complex behaviour need not be the product of a complex control system.

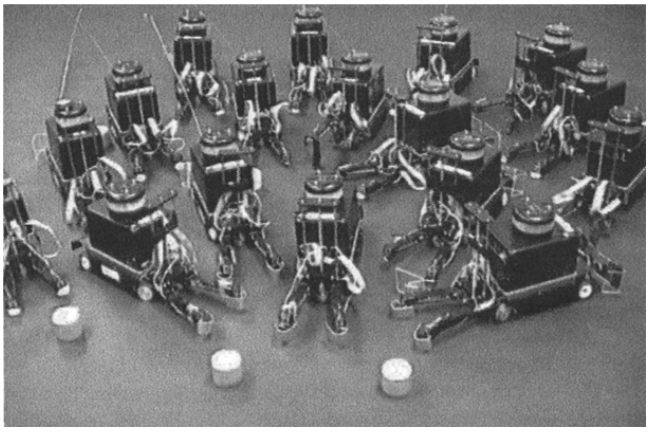

Figure 2.14. Matarić's (1994) "Nerd Herd" of cooperative robots. From Arkin (1998) page 371.

- Intelligence is in the eye of the observer (see also Agre & Chapman (1987)).
- The world is its own best model (further abstraction is not needed).
- Simplicity is a virtue.
- Robots should be cheap.
- Robustness in the presence of noisy or failing sensors is a design goal.
- Planning is just a way of avoiding deciding what to do next.
- All on-board computation is important.
- Systems should be built incrementally.
- No representation. No calibration. No complex computers. No high bandwidth communication.

This system is fundamentally reactive. "... complexity... may be the result of the interaction of simple opportunistic strategies with a complex world." (Agre & Chapman 1987).

Pirjanian (2000) also points out that behavioural solutions are not necessarily optimal, mentioning that achieving "good-enough" solutions have been "... an implicit agenda in behaviour-based robotics all along".

Again, this is closely related to the 'swarm ethos' described in Chapter 1.

The "Nerd Herd" of Matarić (1994) is a widely cited example of the Subsumption approach implemented upon a group of robots (see Figure 2.14). The Nerd Herd's control architecture was based on the belief that "... for each domain a set of behaviours can be found that are *basic* in that they are required for generating other behaviours, as well as being a minimal set the agent needs to reach its goal repertoire" (Matarić 1994). The implication is that there will usually be a tractably small number of basic behaviours.

The idea of basic behaviours corresponds well with eusocial insects: "... the same piece of behaviour can serve different functions in different ants (or even in the same ant)" (Sudd 1970).

The Nerd Herd's basic behaviours; "safe-wandering", "following" "aggregation", "dispersion" and "homing" are employed in various combinations to perform "flocking", "foraging" and "docking" tasks (Matarić 1994). More universal primitive behaviours are available in Huntsberger, Aghazarian, Baumgartner & Schenker (2000), who use hardware-derived basics such as "crab", "turn", "stop", "move", etc. However, the gulf in scope between these primitives and the precision required of the final task necessitates many intermediary linkages. These sources exploit the desired tasks to formulate the base behaviours, they do not place firm limitations on the number of these atomic units – Huntsberger, Pirjanian, Trebi-Ollennu, Das Nayar, Aghazarian, Ganino, Garrett, Joshi & Schenker (2003) implies that the physical structure of their robots drives these primitives (see Huntsberger et al. (2003), Fig 3).

Other examples of loosely similar designs based on finite action-spaces are Fujisawa et al. (2014) and Nouyan et al.'s (2009) Similar ideas can be found in the work of Arkin (1992) and Arkin & Balch (1997), while the much grander 2^{10} strong swarm of Rubenstein et al. (2014) also employs three basic behaviours to create a self-assembly algorithm by linking the three elemental behaviours together "with finite state automation." See also Pagello, DAngelo, Montesello, Garelli & Ferrari (1999), Adouane & Le Fort-Piat (2004), Stroupe, Okon, Robinson, Huntsberger, Aghazarian & Baumgartner (2006) Lötzsch, Bach, Burkhard & Jüngel (2004), Alfredo, Arkin, Cervantes-Perez, Olivares & Corbacho (1998), Kernbach (2013), Ruiz & Uresti (2008), Zhang, Xie, Yu & Wang (2007), Schmickl & Crailsheim (2008*a*) for other behaviour-based examples in a great variety of environments.

Several systems exist for selecting or weighting behaviours. In Brooks's (1999) Subsumption system, layers of finite state machine can suppress other elements (Brooks 1999). Combined outputs by means of summation are discussed in Arkin & Balch (1997) and by Matarić (1994). Parker (1998) employs time-based motivation to activate behavioural sets, while the "motivational behaviours" are allowed to cross-inhibit other motivations. This motivational input may also come from other agents (Parker 1998, Bekey 2005). Similar cross-inhibition is also demonstrated by Werger & Matarić (2000). Both of these systems assume explicit communication between agents.

Finally some, less general projects use "locker-room" agreements based on an exploitation of *a priori* task and environmental knowledge (Kok, Spaan & Vlassis 2005) (Pagello et al. 1999).

2.6.2 Architectures and Swarms

While general cooperative systems have been successfully implemented, they do not usually support the definition of a swarm agent (see Chapter 1).

There are two major problems with the architectures discussed above in relation to swarms:

Not aimed at swarms agents. Previous general architectures are often aimed at more complex robots non-swarm tasks. From a swarm perspective, this leads to several problems:

- *Many behaviour primitives* – The possible robots may have many degrees of freedom and be required to undertake very precise motions. See Huntsberger et al. (2003), Stroupe et al. (2006) and Arkin & Balch (1997). Because there may be great heterogeneity among complex robots the behaviour primitives are often without firm limits, the numbers and type of primitive behaviours are instead deduced from task constraints (see Matarić (1994)).
- *Abstract behaviours* – The many possible configurations of complex robots place considerable responsibility for low-level behaviour implementation in the hands of the robots' designers. The emphasis of the control architecture becomes higher-level decision making – see Parker (1998) and Parker (2008). High-level abstractions can also lead to very long tool-chains (see Arkin & Balch (1997), the complexity of which is inimical to the swarm ethos.

Reliance on explicit communications. Many previous architectures rely heavily upon explicit, direct wire-less communication between agents (Parker 1998), (Huntsberger et al. 2003), (Werger & Matarić 2000), (Isik et al. 2007). Although some systems can function without this (Isik et al. 2007), the concept is considered un-swarm-like, relying upon data-transfer rather than stigmergy and self-organisation.

The data-transfer is often used in behaviour selection, activating or inhibiting behaviours on the basis of *other agent's decisions* (Parker 1998), (Werger & Matarić 2000). The implication of this is that single agents are sufficiently complex to solve tasks on their own. However, swarm agents should be "relatively incapable or inefficient on their own with respect to the task at hand" (Şahin 2005). Redundancy is an integral part of swarm cooperation.

The above sources have all implemented successful cooperative robot groups. However, for swarm agents, these features are undesirable.

Chapter 3

The Swarm Architecture: Design

> . . . and such machines had often been used to break the ice at parties, by making all the molecules in the hostess's undergarments simultaneously leap one foot to the left. . . Respectable physicists said they weren't going to stand for that sort of thing; partly because it was a debasement of science, but *mostly* because they didn't get invited to those sort of parties.
>
> — Douglas Adams,
> *The Hitchhiker's Guide to the Galaxy*

Summary

This chapter describes the architecture developed in this project. The Architecture draws upon the outline of insect swarms discussed previously in its design.

Contents

THE ARCHITECTURE developed in this project created a framework for the control of insect-like swarm robots. The Architecture is designed to:

- Provide the central program in each robot.
- Reduce features that would exclude a given class of swarm-like behaviours (task-agnostic). The Architecture does not exploit task-specific knowledge, although the task is assumed to be amenable to implicit decentralised control.
- Strictly limit the number of behaviour primitives to reduce system complexity.
- Be implemented on any "insect-like" robot without major modifications (hardware-agnostic).
- Provide insect-inspired control, using the concepts of self-organisation and stigmergy to reduce complexity, but not to slavishly reproduce eusocial insect traits.
- Provide cooperation that is not limited by swarm size.

To realise this, several key assumptions were central:

1. Agents are inexpensive and mechanically simple. A swarm-like robot uses the bare minimum of moving parts necessary to move, manipulate objects and perceive its environment.
2. Moving parts are complex and expensive, powerful computers are cheap. While control should be simple, we can assume that relatively high-level computational tools are available.
3. Communications should be implicit, local and obvious. Complex global broadcasts of whatever medium are not consistent with swarms[1].
4. A given hardware design restricts a swarm to a specific broad environment (eg, ground, air, water, space, etc.) and is not suitable for another. This limits the number of tasks possible for a given design.
5. There are a tractably small number of useful object types in each environment (see Section 3.2). While task knowns may not be exploited, environmental knowns can be, as a given machine is limited to one environment in any event.

[1] While some previous swarms have used local communications and Isik et al. (2007) demonstrate that groups can cooperate better in the presence of communications, it was believed that if this Architecture was built on fully silent robots, such communication could be added at a later date – see Section 5.1.2 for a discussion on how this might be done.

6. Individual swarm agents make mistakes. Meaningful results from simple agents is the objective, not maximum efficiency and speed. We expect mistakes and failure.

7. Users want simplicity. The persons controlling the swarm want to be able to do so with the the minimum of fuss. They do not want to learn a new language, complex graphical user interface or multi-step compilation process and the field of swarm robots is too small for such a paradigm to be widely adopted.

3.1 Design Overview

Arkin (1998) quotes Matarić's 1992 definition of an architecture:

> "An architecture provides a principles way of organising a control system. However, in addition to providing structure, it imposes constraints on the way the control problem can be solved."

The Architecture developed in this work[2] provides just such a structure. By exploiting the inherent limitations of swarm agents, The Architecture strives to provide a software basis for swarm agents that allows for many different swarm tasks and insect-like swarm robots to be accommodated.

To achieve this, there are a number of broad features:

- The robot's raw sensor data is reduced to a list of **objects** (see Section 3.2) that are stored in a central location. This "Object List" is the only legal source of information on the robot's external environment.

- Each robot's physical actions are limited to a list of five basic motor schema or **actions**. By drawing on the definition of a swarm agent developed previously we argue that these basic actions are all a non-specialised "worker" agent requires.

- Users couple relevant objects to actions to create **behaviours**. Behaviour creation process is not considered a language, although it has language like features. The creation process is not designed to be extensible (new fundamental features are not expected) and is intended remain simple, in keeping with the swarm ethos.

These features are intended to be run in parallel. Strict order is not important, as long as all are run at sufficient speed to keep pace with real-world events.

There is significant standardisation and little scope for the agent designer to add non-standard embellishments and only a narrow window for user interaction[3].

The Architecture processes are:

[2]The design does not have a name, in all notes and subsequent writings the author just referred to it as "The Architecture", which, in the author's opinion, ultimately sounds better than any contrived phonetic acronym.

[3]In this chapter, flow-charts indicate where low-level implementation is unimportant, provided procedure is followed, pseudo-code indicates where algorithms are important, but code-implementation non-critical and plain-C code indicates that low-level data-types and function input/output is critical. The level of abstraction used indicates the degree of designer freedom.

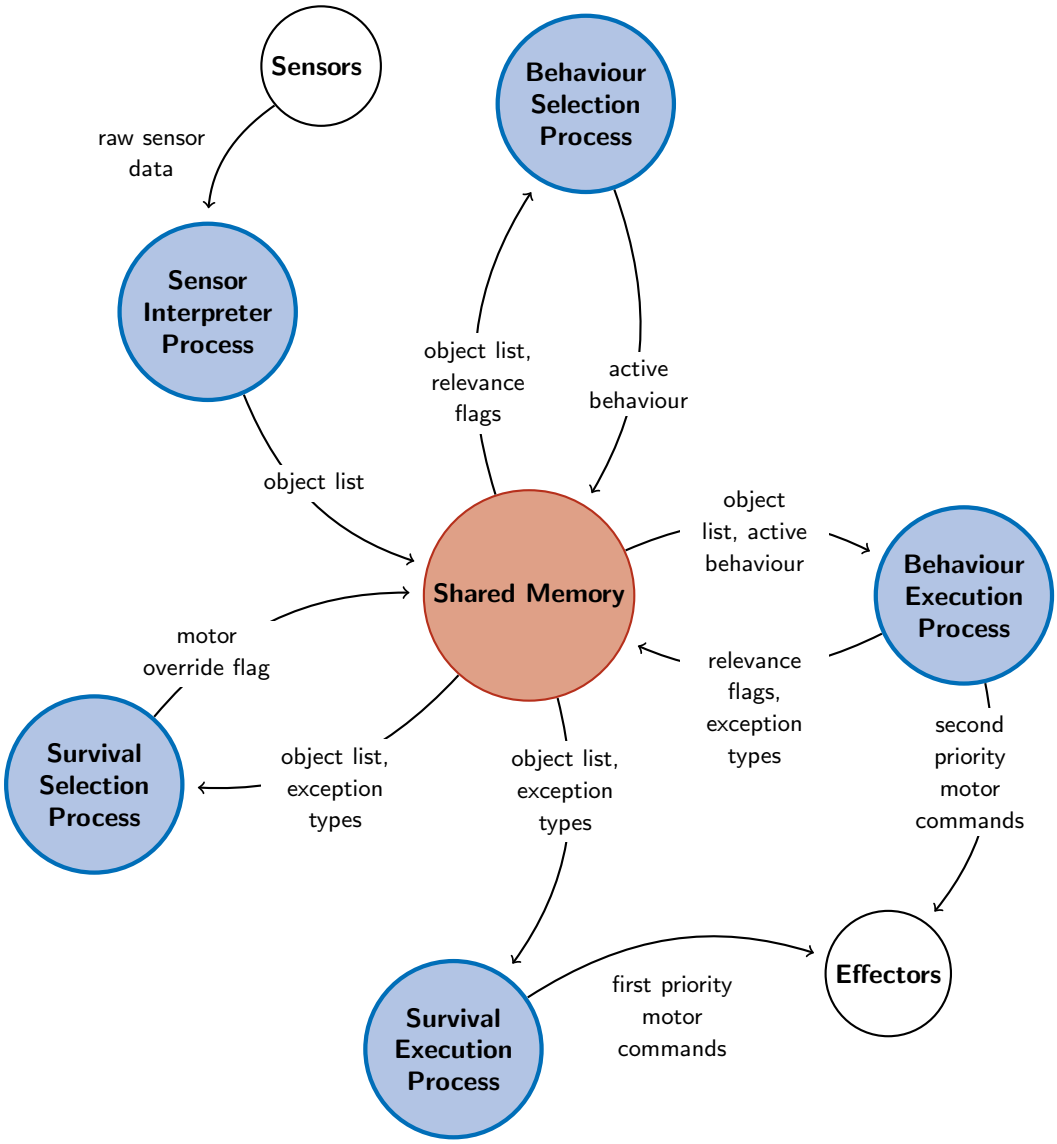

Figure 3.1. The complete Architecture showing all processes and their interactions. The blue elements represent separate, parallel processes, with the shared memory in the centre providing inter-process communications. Note: any method of inter-process communication will suffice, specifically using shared memory is not critical.

sensor interpreter – accesses the raw sensor data and updates the object list.

behaviour selection – decides current active behaviour.

behaviour execution – calls the currently active behaviour.

survival selection – decides upon the need to execute survival actions, such as avoidance.

survival execution – calls currently active survival process (if any).

See Figure 3.1 for the interrelation of these processes. Shared memory is used to transfer information between processes. The exact form of inter-process communication is not critical, threads could also have been used.

3.1.1 Parallelism and Computing Requirements

The Architecture is intended to run on small, single-board computers with UNIX-like Operating Systems[4]. The Architecture is not intended to be firmware on a micro-controller.

The operating system need not be real-time in the strict sense, but there should be sufficient computational resources that The Architecture sub-processes run "fast enough" to keep approximate pace with physical events outside the robot.

As the robots' perception system relies heavily upon computer vision, robots' computers must be powerful enough to recognise objects visually with no noticeable lag between object appearance and visual identification. In practice, this means that there must be sufficient memory to hold multiple frames and a clock-speed commensurate with processing many frames per-second. The author found that 256MB and 700MHz (respectively) are barely adequate for five frames per second (see Appendix B for more information).

The Architecture's component processes communicate via a POSIX shared memory structure. Read and write race conditions are prevented by use of POSIX semaphores. The shared memory data structure is small and it is assumed that any processes read/write duration is very short.

The Architecture does not enforce an ordered sequence among its sub-processes, instead it is assumed that all processes are running fast enough that all shared information is keeping approximate pace with physical events.

The Architecture simply uses the default Linux process scheduler to assign computational resources. In practice it was found that only the sensor processor (vision system) imposed any significant computational load. All other processes were negligible by comparison (using an order of magnitude less system resources).

As an aside we should note that the simulated version of The Architecture (see Chapter 4) is implemented in a sequential form that could be ported to

[4]Recent developments in high-performance, ultra-low-cost, small-scale computers (see Appendix B) leads the author to respectfully disagree with Brooks's (1999) assertion that behaviour based robots should have "No complex computers". As these machines are now readily avaliable, computationally complex tools such as vision are now accessable by small, inexpensive agents. It is the author's belief that all available tools should be used to expedite design.

a more limited micro-controller. However, this is an artefact of the simulator's step process and is not an intended design direction. The author believes that, given the extreme cheapness of single-board Linux computers, developing a system for tiny micro-controllers represents a technological dead-end.

3.1.2 The Designer and User

The Architecture strives to maintain a sharp distinction between the designer and the user. The designer is the individual or organisation who designs the individual robot agents. It is assumed that a given swarm machine will be designed with a broad, but limited environmental and task range in mind. For example: it may be a basic "worker" unit designed to manipulate objects on a two dimensional ground. Regardless of how broad the intentions of the governing architecture, the designer will introduce inherent limits.

In regards to this Architecture, it is the designer's responsibility to produce the agent's mechanical and electrical systems and implement The Architecture framework upon them. Low level sensor interpretation (see Section 3.2), action functions (Section 3.3) and Architecture basics are all the designer's responsibility. The designer should produce a mechanically, electrically and Architecturally functional *tabula rasa* suitable for a given environment and wide variety of tasks – exactly as personal computers and operating systems are created.

The user is the second party, who may have no connection with the designer and is responsible for applying the designer's work to real-world tasks. The user is expected to understand The Architecture as a concept, but not to have any knowledge (or real interest) in the minutiae of the implementation on their unit – exactly as a person may know how to work their personal computer and understand operating system concepts, but may not understand (or care about) the kernel's source code. The user is expected to create behaviours suitable to the desired task according to The Architecture's strict laws (Section 3.4).

3.2 The Object List

How an agent senses its environment is fundamental to its operation and is central to The Architecture. What it is required to see, is dependant on the agent's physical construction and task parameters, respectively. Regardless of how general an architecture is, these are still unknowns that cannot be accounted for in The Architecture design without exploiting *a priori* task and hardware knowledge. Thus, these unknowns should not influence the decision-making process. The Object List provides the necessary isolation between perception and decision making.

In The Architecture, the only legal source of information about the environment *outside*[5] the robot is via the object list. A dedicated process is responsible for accessing all external sensors and reducing the data to entries in the list. Each entry defines at minimum the object type, its position relative to the robot and the robot's confidence in its existence (see below). Designers may add whatever else they think will help them, but these quantities are mandatory. This

[5]Interior information, such as inertial sensors, odometers, clocks, etc. that describe the robot's internal state are a separate case and will be addressed in Section 3.3 and in more depth in Section 5.1.3.

sensor interpreter process is the *perception* component of the agent.

As the Object List is the only legal source of external data, it is required by all other processes in the system and is implemented as a shared memory structure, protected from race conditions by semaphores.

How the sensor interpreter accesses and processes sensor data is irrelevant to The Architecture and is a matter for the designer. The only constraints are that the object data must accurately identify the object, its position relative to the robot and be updated fast enough to keep approximate track of its motion.

Each object contains the following data:

- The object type. This is an `int` flag. Objects are not defined with any finer detail. Like Agre & Chapman (1987) objects are "not defined in terms of specific individuals".
- The object's range relative to the camera. This is measured in metres from the camera and is stored as a `float`.
- The object's bearing left and right relative to the camera (left is negative, dead ahead is zero). This is measured in radians and stored as a `float`[6]
- The object's confidence (see below). This is a value between 0 and 1 and is recorded as a `float`.
- The object's temporal duration in seconds, employed in object tracking, projection (see Appendix B.1.4) and confidence calculation (see Section 3.2.2) and is recorded as a `float`.
- The quality of recognition match. This is a value between 0 and 1 and is recorded as a `float` (see Section 3.2.2 and Appendix B.1.4).

When implemented, the object list is simply a shared structure containing these quantities. Each of these quantities is an array of size $1 \times$OBJ_MAX (50 in the real-world or 100 in the simulations). The actual method of object list implementation is unimportant as long as the above quantities are available at high speed.

3.2.1 Object Types

By making raw sensor data interpretation the responsibility of the designer, the number and range of object types is of necessity defined *a priori*. This is a limitation on generality, however it is not as significant as it first seems. A given agent design inherently limits the swarm to a relatively narrow range of environments and tasks; a underwater robot cannot function in space, a drilling robot will be of limited use in a cleaning task and so on. This is a fundamental limiting factor discussed previously; an agent's physical form is defined by its environment and any given environment is presumed to have a limited number of interesting object types.

[6]In practice, it was found best to limit scope of the perception system. Robots with a very wide field-of-view found too many potential obstacles and spent much time avoiding. By contrast robots with too narrow a field-of-view recognised too few objects for decisions to be made.

Type ranges can be very broad, but some objects are obviously mandatory. All agents would need to recognise other agents, for example. Object type descriptions can be very general – "round object", "big object", etc. – but must be unambiguous (see Section 5.1.3 for more on general type implementations). It is basically assumed that the designer would use a camera as the primary sensor, possibly supported by simpler devices. Such as infrared proximity sensors.

In both the simulation and real-world incarnations of the architecture, the following objects were defined and represent a basic minimum for The Architecture:

target-object
: Any object that the robot would have to manipulate or interact with. For some of the tasks there were several types of target object and associated types[7]. These other targets are labelled `TARGET_OBJ_1,2,3...n`. The designer may implement as many of these as the sensors can discriminate.

target-object-group
: Collections of target objects in physical proximity. Proximity was established by means of the cosine rule and comparison to a limiting distance. Experience shows that a description of semi-completed structures is necessary to allow for grouping, building or similar activities (see Section 4.3.4 for more on this). Objects of this type were also represented in the object list as individual target objects.

  As we shall see, the real-world objects are represented by repeating patterns, meaning that a physical object might be (and usually was) detected more than once. To prevent individual objects being falsely identified as groups, the real-world implementation defined a group as three or more instances of an object in proximity.

captured-object
: An object that is within the robot's control. In practice, a captured-object is anything that trips the infrared sensor attached to the scoop (see Appendix B). There should be as many captured object types as there are target object types, but in practice, it was found difficult to distinguish between different captured object types and thus a single general capture type was implemented. This was found acceptable for the tasks attempted. See Section 5.1.3 for a discussion on a more general version. Objects of this type were not recognised as separate target objects.

capture-by-other
: An object under the control of another agent. This type was found to be necessary to prevent agents from trying to steal objects from each other. In practice this is simply any target object that was sufficiently close to another agent. There were as many captured by other object types as there are target object types. Objects of this type were not recognised as separate target objects.

[7] For historical reasons, the target object is sometimes referred to in commentary as the "ball". The original task was "ball" passing and the name stuck, even though the final target looked nothing like a ball.

agent

Any other agent. Agents were not identified as individuals, so this was anything bearing "agent" markings. Notice that it is the designation of agent that is important. Anything the designer fancies may be designated another agent, provided it is consistent with the assumption that it is an animate swarm member, working towards the swarm's greater goals and can be recognised as such by the rest of the swarm. Thus "agent" may be any variety of robotic or even non-robotic agents. Agents do not reason about their "kin" or receive explicit instructions from them (as in Werger & Matarić (2000)), they are simply another object.

proximity-object

An object that tripped the robot's low-level proximity sensors. These objects could (although usually were not) be recognised as other object types, if they were within visual range.

In all cases, the object type was identified by a unique integer flag.

Once again designers can include whatever other types they think may be useful, but our experience suggests that the above comprise a practical minimum. However, designers should make their sensor interpreters and type definitions modular, support for extensions should be comprehensive. More ideas on expanding object recognition can be found in Chapter 5.

For a detailed discussion on how these objects were recognised in this project, sensor development and associated problems, see Appendix B.1.4.

3.2.2 Confidence

Along with type identification and positional data, each object in the list is required to have a "confidence" value. As real-world sensors tend to be noisy and unreliable, false positives and negatives are possible (experience would suggest common). As objects are integral to the decision making process, ignoring sensor uncertainty results in unstable decision making. Thus by convention, all aspects of The Architecture that have to deal with discrete objects (rather than types), make their decisions and calculations based on confidence, rather than counting instances of the relevant type.

See section B.1.4 for details on how confidence was implemented in the real-world and simulated architecture.

3.2.3 Insect Sensor Comparisons

We should pause here briefly and compare the sensory system as can be produced with a CCD camera to the sensory perception of social insects, covered in Section 2.1. As mentioned, insects such as ants and termites have limited vision and rely heavily on touch and chemical detection.

Touch type sensors exist for robots in many forms, but all are effectively limited to detecting an "object" in a Boolean sense. To classify objects by this type is not, by itself, sufficient. In our implementation we include proximity objects to cover this type of sensor and employ them in both the hardware and simulated robots.

Chemical analogue sensors have been implemented in robots, both as virtual pheromones (Payton et al. 2005) and as literal chemical secretions (usually alcohol applied to a flat surface with a pen) (Kallel et al. 2008).

The former was considered an inappropriate starting point as it would imply some means of explicit global or peer-to-peer communication. It was believed that The Architecture should not rely on such methods from the beginning, instead using them to enhance an existing design – see Chapter 5 for more on this.

Literal chemical markers were considered and discarded as an overly specific requirement. Although the hardware exists, it is not common and has not had the development lavished upon cameras. Cameras have recently become cheap and commonplace and are highly suitable as an object classification sensor. The author does not believe that chemical detection technology exists in a comparable form.

Nevertheless, a computer vision system does provide some of the features of an ant or termite's chemical perception. It is a passive, agent centred perception system, needing no global information or actions by other agents. It can recognise objects as classes, often at some range. It lacks the "field effect" of odour-based chemical systems – it is generally not practical to lay down a visual trail in the same sense that an ant may deploy a chemical one. Visual recognition is thus limited in temporal scope. Likewise, perception cannot be reinforced the way repeated path use can be reinforced among ants. The visual robot has no easy means of judging how many agents have used a path in the past.

By accepting these constraints, the visually controlled robots became more reactive in their perception than their biological counterparts and they must have line-of-sight to an object at some point in that object's life for detection to occur.

However, assuming that some functional analogue of insect chemical markers could be built (or has escaped the author's attention), it could be easily integrated into the Object List concept. A trail left by another agent simply becomes another instance of the AGENT object type, with the confidence denoting the strength of the trail. The physical position of the AGENT object instance created by this trail would shift as the observing robot moved along the path, always staying ahead of observer until the trail ends. As mentioned, ant trails are non-directional, thus the robots would not need to know which way the trail laying robot travelled on its path.

More static chemical markers – such as those used by termites to mark the soil pellets in their construction – are here considered an extension of type-based visual recognition. Instead of using their eyes, the termites use their olfactory sensors to distinguish object type.

3.3 Actions

The atomic unit of the behavioural side of The Architecture is the **action**. An action is a single function that controls the robot towards one, simple goal and is the direct link between the Object List to the motors. The Architecture exploits the expected features of a swarm agent to reduce the number of supported actions to five.

An action is the direct and only link between the Object List and the effectors. They are reactive functions and have one goal that focuses on controlling the robot relative to the physical location of the "best instance" of a single object type.

In The Architecture, we make a number of assumptions about robot hardware, based on the perception, manipulation and locomotion triple and the swarm ethos:

Sensors Fixed to Robot
A robot's sensors have no independent articulation, to change the sensor focus the robot must physically move. This is because such a machine can "look around" without needing additional motors. Because the individual robots should be as mechanically simple as possible it follows that if they can sense without additional equipment, then they should.

Each Robot Has One Manipulator
While it is necessary for each robot to possess a manipulator it is assumed that they only need one. It is further assumed that this one gripper is limited in motion, that it only grips and is not independently articulated. To move the gripper, the robot should have to move its body. This is an extension of the fixed sensors assumption.

The Robot can Control its Position
The motors can move the robot relative to other perceived objects.

Creating actions is the responsibility of the designer and while the designer may use whatever internal source of information they deem necessary, their only legal source of external information is via the Object List. It is forbidden to access external sensor data directly from actions.

We do not rely heavily on internal sensors in either the simulated or real-world implementations of The Architecture (aside from the system clock). This was because both versions were implemented on fairly simple machines in straightforward environments. More complex implementations with better object tracking or more sophisticated locomotion may wish to employ inertial sensors, odometers, etc. Section 5.1.3 discusses how this may be done within the requirements of The Architecture.

Previous group and swarm architectures have also used small libraries of primitive behaviours to construct more complex patterns (for example: Matarić (1994), Arkin & Balch (1997), Arkin (1992) etc.), but this Architecture sharply constrains the number of primitives based upon the expected features of a swarm agent. If the agents' effectors are restricted to manipulation and locomotion and perception formalised as objects, then only four main primitives present themselves:

Move To Object
Move the robot from its current location to the best instance of object type T. An optional range minimum may be passed as an argument. Upon the successful completion of this action, the object should be at the specified range from the robot and the robot should be facing the object.

Move From Object
Corollary of the move to action; remove the robot from the best instance of

object type T. An optional range minimum may be passed as an argument. Upon successful completion of this action, the object should be at least the specified range away from the robot and the robot should not be facing the object. These two actions represent the locomotion aspect of a swarm agent.

Grasp Object
Capture the best instance of object type T with the robot's manipulators. Upon successful completion of this action, the object should be as fully under the physical control of the agent as practical.

Drop Object
Corollary of the grasp object action. In this case the target object type is implicit, it is whatever is currently grasped. Upon the successful completion of this action the object should be fully outside the robot's physical influence and (as much as possible) outside its perception. These two actions represent the manipulation aspect of the swarm agent.

Each of these actions performs its objective and that objective only. Other considerations such as avoidance are the responsibility of survival actions (see below). Nor do we advocate the vector summation of Arkin & Balch (1997). Each of the four actions above is a complete function for the physical output of a swarm agent and is expected to have its own control loop and command the effectors in a winner takes all fashion.

How the four actions complete their goals is the responsibility of the robot's designer, who must integrate them with their physical design. However, there are some minimum constraints:

- All internal action loops must be sensitive to The Architecture shutdown flag, which is stored in shared memory and is accessed via the `get_shutdown_flag()` method. All loops must be broken out of if The Architecture is shutting down.

- All internal action loops must be sensitive to the current active behaviour. Actions are called via behaviours (see below) and terminate if their parent behaviour changes. The current active behaviour index is stored in shared memory and is accessed via the `get_active_b()` method. The action must be passed the index of its calling behaviour as an argument.

  However, only the locomotion actions must terminate immediately if the behaviour changes. Grasp and drop are both permitted to complete their motions and are thus partially blocking (see below).

- If an action achieves its goal, or terminates due to active behaviour change or Architecture shutdown it returns SUCCESS. If its target object type disappears from the object list, or there is a technical programmatic error, it returns FAIL. Upon the receipt of a FAIL the calling behaviour will terminate (see below).

- Actions are program functions and have the following prototype:

```
int action(  shared_memory* shm,
             int obj_type,
             int behave_id,
             float min_range );
```

However, some of these inputs are implicit: in the grasp and drop actions the minimum range is implied by the robot's manipulator dimensions. The target object type of the drop action will be whatever is in the manipulators at that time. The details of actual implementation in both simulation and hardware are discussed in Appendix B.

Notice that the input into the action functions is an object type, not an individual instance. Actions must follow a standard for selecting which instance of that type they focus on. This is achieved through the `get_active_obj()` shared memory method and is based on object type and confidence – the object of the specified type of the highest confidence is defined as "active" by Architecture-wide convention.

As all actions are implicitly or explicitly focused on a single object, a global position and navigation system is unnecessary, only the location of the object relative to the robot is significant. Swarm agents are presumed to be reactive and respond only to concrete objects they can perceive. The Architecture does not support abstract movements, all agent motion must be *object-orientated.*

Four fairly simple commands may seem little enough with which to construct a coordinated swarm of unbounded size, but consider again the generalisation of eusocial insects in Section 2.1. Like Matarić (1994) we believe that for each domain, a *basic* set of actions exist. However, we define the domain as *only* tasks that a perceive-move-manipulate swarm agent is capable of and limit the basic action set by considering the limitations of such an agent and enforcing the use of the object list.

The reader may notice some similarity between this architecture's "actions" and the motor "schema" discussed in Section 2.6. They represent the same concept, but here we have exploited the limitations of swarm agents to strictly control the number and content of the individual schema. Arkin's (1998) motor schema idea is a broader one than that used here. The approach described in Arkin & Balch (1997) and Arkin (1998) creates a design feature that could be employed in *any* machine, not just in a swarm agent. To control their robots they create a schema for every possible action, we are not interested in duplicating that generality. Swarm robots should not be able to perform *every* possible action, they are limited as individuals by the requirements of the swarm concept. Thus the actions used in this project could be viewed as a subset of the larger "schema" set.

3.3.1 Blocking Actions

If an action causes an object to change state (eg: "drop" causes an object to transition from "captured-object" to "object"), it should complete before the next action is executed. This is to ensure that the object state transition is complete, once the object changes type, new behaviours may become relevant. If the system reacts too fast, a new behaviour may be implemented before the object is fully gripped or dropped.

To prevent this from occurring, the "grasp", "un-grasp" and "avoid" actions should wait for several seconds before returning from a successful execution. This wait is most easily implemented by a simple loop that cycles until `timeout` seconds have elapsed. The exact mechanism is unimportant as long as the behaviour execution process is unable to call new behaviours for a small period.

These blocking actions have a small impact on the robot's reaction speed, but it is considered that tasks that require highly a precise, rapid response in less than a few seconds are not swarm tasks.

Originally, all actions were required to be sensitive to behaviour termination. If the calling behaviour ceased to be the active behaviour, then its current child action would also be required to terminate, regardless of other considerations.

The trouble with this approach is that it encouraged poor performance. A robot that engaged the drop action was supposed to turn from the dropped object, once the object left the robot's control, to prevent it from being re-acquired. However, as soon as the object left the robot's control, the captured-object type no longer existed, causing that behaviour's relevance to crash to zero. Thus the drop action was never properly completed.

Once the drop action execution became blocking, the dip in relevance became inconsequential, as the relevance conditions would be re-set by the next action block to match the changed environmental conditions.

It was found that "drop/un-grasp", "grasp" and "avoid" should be made blocking for this reason.

Blocking actions were introduced piecemeal over the testing process, with both the avoidance and drop actions having their sensitivity to behaviour termination removed. There was little observable affect on macro swarm behaviour, as the motions of other agents and the inaccuracies in individuals do not allow stop-start loops to run for long. There is however, a observable improvement in small areas of individual performance.

3.3.2 Agent Survival

As the reader may have observed, the four behaviour actions above do not describe how a robot is to protect itself against collisions, power loss and other disasters. These difficulties are handled by the "survival actions" that are summoned and selected by separate processes that operate in parallel to the base behaviour actions (see Figure 3.1). Survival action outputs commands with a higher priority in motor control than the four behaviour actions, allowing the robot to protect itself.

Avoidance is the most obvious omission in the base behaviour actions and the only one corrected in the actual implementation of The Architecture (although the potential for others exists – see Section 5.1.5). The base behaviour actions do not require that the control loop considers anything more than the target object, the termination criteria (if any) and the technical flags as described above. Given this single-mindedness, the behaviour actions would be quite content to drive the robot into or through any object that comes between the agent and the target object.

It would have been possible to specify that the action functions must consider avoidance in the specifications, however this would imply that each action had two objectives. If the user wished to consider alternative behaviour when agent battery power ran low, that would necessitate base actions with three objectives, and so on. Clearly, if every special case were to be included into the action functions, they would swiftly become overly complex and defeat their purpose.

A survival action has one function. As there is expected to be only a few (less than four), they are hard-coded into The Architecture's survival selection and execution processes. The former checks a simple criterion to see if the survival

action is necessary, the latter executes any necessary survival actions. As the implementation version only used "avoid", there was no need to discriminate between survival actions (see Section 5.1.5 for how to do this in future versions). If a survival action is necessary, a flag is set and that action function is called by the survival execution process. Survival actions are stand-alone units, there is no "survival behaviours". Recall that according to the swarm ethos, individuals are expendable and survival execution does not have to be perfect in all situations.

Survival actions have a higher priority in their control of the motors than the regular behavioural actions. When a survival action is necessary (see Section 3.4.6) a `motor_override` flag is set. In the implementation of The Architecture used in this project, the motor control function discriminated between input from the regular behaviour actions and the survival actions on the basis of this flag. If the `motor_override` flag is set, only the output from the survival actions is considered, while output from the regular behaviour actions is jettisoned. Notice that the enacting of a survival action does not affect the execution of regular behaviours, it simply causes their motor commands to be ignored.

3.4 Behaviours

A behaviour is a sequence of actions that need to be executed sequentially to physically alter the state of either the agent or the environment in order to achieve a *single* task goal.

Behaviours are the building blocks of The Architecture. It is through designing behaviours that the user is able to apply the swarm to new tasks. The behaviour selection governs what an individual robot actually does and the relevance of its constituent behaviours determine when this is done.

To do this correctly, it was necessary to narrowly define what is possible within a behaviour – if The Architecture is to be truly general, the user should not need to give too many explicit instructions. To this end, behaviours are constructed from discrete "action blocks" that are executed in strict sequential order and contain everything necessary for the behaviour at that point – all the relevant objects, which objects to avoid and which actions to perform on which objects. If any action block fails (ie, its action returns a FAIL code) then that behaviour is terminated.

Behaviours are selected via a modification of the insect model described in Theraulaz et al. (1998). This mechanism is a combination of an internal time variable and external stimulus in the form of the presence or absence of relevant objects.

The entire behaviour process is intended to be as close to full automation as possible, requiring the least input from the user and constraining such as necessary to a formulaic structure that could ultimately be machine written.

3.4.1 Behaviour Selection

Behaviour selection is a separate process that runs constantly and in parallel to the execution process. The selection algorithm was inspired by the method of Theraulaz et al. (1998) – see Section 2.4, but ultimately evolved into a distinct method under the pressures of implementation.

It should be stressed that The Architecture is for swarm robot control. Although inspiration and design elements are drawn from biological sources, the direct replication of biological systems is not the principle objective.

Thresholds

Equation 2.1 provides a means via which behaviours may be selected in an insect-inspired fashion. However, whilst it is relatively easy to implement the self-reinforcing threshold (θ), realising the stimulus proved to be much more challenging.

In Theraulaz et al.'s (1998) model, the stimulus for each behaviour, like the threshold, is time driven. Originally, The Architecture implemented a similar system, but struggled to reconcile this with behaviour relevance.

Originally, it was believed that a behaviour that relied on object type T could not be implemented unless an instance of object T existed in the object list. It was assumed that behaviour selection was a process of making a decision among possible behaviours, with a distinct search or "null" behaviour comprising a fall-back.

To implement this, the threshold model (Equation 2.1) was adapted by removing stimulus and replacing it with relevance; an evaluation of how relevant a behaviour is based on that behaviour's focus and the contents of the object list at that time (see below).

The following equation combined these ideas and provided the basis for behaviour selection:

$$P_{i,j} = \frac{rel_{i,j}}{rel_{i,j} + \theta_{i,j}} \tag{3.1}$$

Where:

$P_{i,j}$ The current ranking for robot i, behaviour j.
$rel_{i,j}$ The relevance of behaviour j to robot i.
$\theta_{i,j}$ The threshold for behaviour j on robot i

The relevance value computed from the presence or absence of objects in the object list. The relevance for a behaviour will be different for each robot, as relevance is calculated based on that robot's observations only (see below).

Each behaviour has a threshold (θ), which is incremented if the behaviour is inactive and decremented if the behaviour is active. Thresholds are floating point numbers constrained between zero and one and are initialised to 0.5. Originally a random initialisation was used, but this was abandoned during debugging in an effort to reduce the number of differences between robots. Thresholds are incremented and decremented at the same rate (see Table 3.4.1).

The behaviour with the highest value of P is the current active behaviour. Unlike Theraulaz et al.'s (1998) system this is not a probability – Section 5.1.1 discusses this difference in more detail.

If two or more behaviours have the same P value, then a random choice is made. Relevance is computed from criteria set by the behaviours (see below). The behaviour selection process adjusts the threshold, assesses behaviour relevance, computes P and sets the active behaviour flag in an endless cycle. It

Quantity	Simulation Value	Real-World Value
behave time (bt)	20 seconds	30 seconds
wander time	10 seconds	5 seconds
threshold change	$\frac{1}{2*bt}$ per second	$\frac{1}{4*bt}$ per second

Table 3.1. Behaviour selection constants in both the real-world and the simulation.

is assumed that this can be done "fast enough" to keep pace with real-world events.

The Architecture also incorporates a fail-safe procedure to ensure that a behaviour is not locked in operation. This is to prevent a robot from continually striving to achieve a behaviour that is impossible (for whatever reason), despite being relevant. A behaviour may only run continuously for BEHAVE_TIME seconds before being terminated. Once this has occurred the null behaviour (see below) is forcibly activated for WANDER_TIME seconds so the robot may move through the environment and obtain fresh sensory input.

This cap on behaviour run-time was found to be important. Remember, swarm robots are not prescient. It is possible (usually highly probable) that their sensors are often wrong and the object list is an incomplete guide to the world. What may appear rational to the termite-like robot could be very different when viewed from the Olympian perspective of the human user. Mandatory behaviour termination ensures that robots do not get stuck in corners (for long) attempting to do impossible tasks on sketchy information.

Such behaviour termination can also mean that individual robots abandon half-completed tasks because of an arbitrary time-out. This is not considered important as it is assumed that some other member of the swarm will be along presently to complete them.

In implementation, the threshold system proved less significant in decision making than the relevance value for each behaviour (see Section 4.5). For this reason we shall not hold the differences between the real-world and the simulation to be any more significant than the physical differences between the robots. Basically, they represent a known bug.

Relevance

Regardless of the condition of a behaviour's threshold, it makes little or no sense to engage it, if it is not relevant. A behaviour that focuses on object type T must have a case of object T to be useful.

When a behaviour is defined, the user selects which object types are needed for that behaviour to be relevant (see Section 3.4.3 below). The user may define any number of object types as relevant and may specify their relationship as: EXIST, OR_EXIST or NOT_EXIST. Only one of these flags may be used per object type, but object types may be joined in any and all combinations. Users define relevant object types at each stage in the behaviour (see Figure 3.2).

These existence flags enable The Architecture to form a simple sentence of sorts for each behaviour step, for example: behaviour n defines object types a

and b as relevant and flags them as EXIST and NOT_EXIST respectively. This yields:

> For behaviour n to be relevant, object type a must exist and object b must not exist.

A practical example of this would be a "pick-up" or "acquire object" action, the target object must exist and the captured object must not exist.

This arrangement is relatively simple and, in an ideal world, could be implemented very swiftly as simple Boolean statements. However, real-world implementations have uncertainty and to directly implement these statements in their simplest form would create great instability as false positives and negatives of short duration add noise to the Object List.

To avoid this a more complex logic was implemented, based around the object confidences. Objects existence is checked using a floating point value between zero and one, not simple Boolean exist/not-exist. When confidences need to be combined (in EXIST, OR_EXIST or NOT_EXIST patterns) `max` and `min` operators are used. These terms are borrowed from fuzzy-logic (see Terano, Asai & Sugeno (1992), Bojadziev & Bojadziev (1995), etc.) but confidence does not imply that an object is a member of more than one type.

The algorithm for this process is described in Algorithm 1.

Algorithm 1: Calculating behaviour relevance using floating point object confidence values. As the confidence is a `float`, the minimum value was used as a logic AND and the maximum value a logic OR. The return value is the relevance of this behaviour and is between 0 and 1.

Data: relevance_list — a list of the relevant object types for this behaviour and their existence criterion (`EXIST, OR_EXIST, NOT_EXIST`).
Data: object_list — all the objects currently in view.
Result: relevance — the relevance of this behaviour

```
begin
    for t ∈ relevance_list do
        conf_t = max ( confidences of all instances of type t );
    exist = min (conf [types flagged EXIST]);
    or_exist = max (conf [types flagged OR_EXIST]);
    not_exist = max (conf [types flagged NOT_EXIST]);
    relevance = min ( (1-not_exist), max (exist,or_exist) );
    return relevance;
```

The fundamental assumption of the relevance/threshold behaviour selection system is that a robot's information is *local*. The relevance algorithm does not consider the location of objects in its calculations and hence will consider all objects in the object list that are of a type flagged as relevant by the behaviour. To prevent the robot from being overwhelmed by objects from across the arena, it is assumed that the limitations of the agents' point of view and sensors will restrict detection.

The lack of location consideration is intentional. Location was omitted from relevance computation for the sake of simplicity and is an exploitation of the

expected limitations of swarm sensors. However it has implications for the future expansion of The Architecture, as discussed in Section 5.1.2.

3.4.2 The Null Behaviour and Zeroth Object

When designing behaviours, there was one reoccurring behaviour that always gave trouble; all tasks need to search. Agents obviously needed the ability to look for relevant objects and even with perfectly idealised sensors occlusion and movement would ensure that there would be times when the agent could see nothing to do.

Unfortunately, it proved quite difficult to fit search into the behaviour pattern. This was because of the requirement that all behaviours be comprised of actions that focused on one object. The trouble with a search behaviour is that it requires an action to focus on a non-existent object, which simply did not fit the object list concept.

However, a search function was essential. To implement the random walk pattern seen in earlier swarm searching projects without violating the fundamentals of The Architecture, an imaginary object was created. Entry zero in the Object List was reserved for the "zeroth object"[8]. This object has no physical existence, but is always at index zero in the Object List and recorded as a fixed distance from the agent with a variable bearing. The zeroth bearing was permitted to wander in a random fashion[9], in effect creating an object on a perpetual random walk.

The creation of the zeroth object allowed all "search" behaviours to be depreciated. Instead, a null behaviour was created. The null behaviour had only one action, `move_to_obj()`, which focused on the zeroth object causing the robot to wander randomly as it steered towards the imaginary object.

The null behaviour had a fixed relevance of zero and threshold of one. It was not included in probability calculations. Instead it was only invoked if all other behaviour had a probability of zero. Thus a search behaviour was always present, needed no special actions and no object focus.

The bearing of the zeroth object was controlled from the sensor process, as though it was another object that could be detected via sensors. Every five seconds (both real-world and simulation), a random number was selected. If this number was less than or equal to half the maximum random range, then the bearing of the zeroth object was set on the left of the robot, if greater it was set to the right. The range of the zeroth object was always fixed at two metres and the confidence at one. The zeroth object was permanently exempt from avoidance.

3.4.3 Composition – Building Behaviours

While behaviours were intended as the user's interface into the architecture, they are closely constrained. The behaviour function structure was built with firm rules and an eye towards eventual automatic compilation.

[8] Interestingly, in Brooks's (1999) original Subsumption architecture, the "Zeroth competency" was avoidance, not search. The similarity of names in this case is coincidental.

[9] When random numbers were used in this architecture true randomness (or an accurate pseudo-random approximation) was not usually needed. Thus not much effort was expended in creating or checking the random distribution. The `stdlib` *C* function `rand()` was used throughout, seeded from the clock.

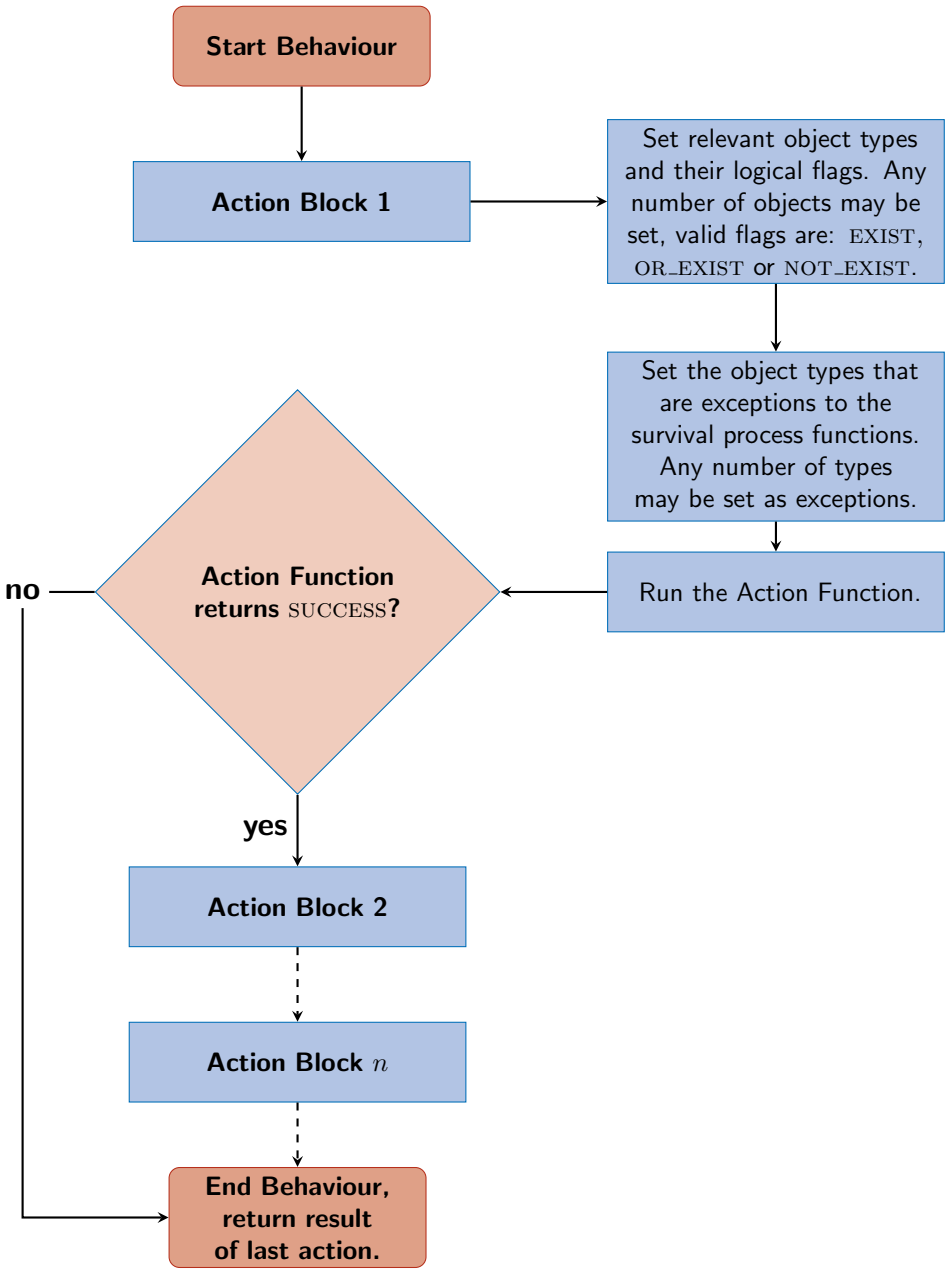

Figure 3.2. The behaviour action block design, showing the first action block.

Each behaviour takes the form of a series of actions in the form of **action blocks**. These are sections of code in a standard sequence that set up everything necessary for a given action. Support for multiple actions was needed as many behaviours are too complex to be expressed as one action, for instance an "acquire" behaviour needs to move to an object, then grasp the object, which is two distinct action calls.

There was no formal limit to the number of action blocks in any behaviour, but experience showed that overly complex behaviours comprising many action blocks were both unstable and unnecessary.

An action block is comprised of the following steps:

Setting Relevant Objects
: Which objects are relevant for this part of the behaviour and what are their relationship. There may be any number of these in whatever relationship the user desires. Again, experience showed that simple was best.

  For readers interested in the code, it is important to note that for the first block, this section is repeatedly called by the behaviour selection process to determine behaviour relevance. Thus in the code this section is protected by an if statement to ensure that the selection process does not execute the behaviour. This is a coding convenience and has no relationship with architecture functionality.

Exception Objects
: Which object types are to be ignored by survival functions (see Section 3.3.2 below). These can be all of the object types or none and is necessary to ensure that behaviours that involve close proximity to objects (such as those involving "grasp") do not trigger avoidance.

Actions
: Call the action function. Action arguments are a pointer to shared memory, the relevant object type (if necessary), a calling behaviour identifier and an optional conditional for-loop termination (if relevant).

This process is illustrated in Figure 3.2.

Once the action function has been called, it will proceed until the target object no longer exists, the loop termination criterion has been achieved, the active behaviour changes (unless blocking) or The Architecture shuts down. All of these things are checked each time the action loop cycles to prevent processes lock-up and to allow clean shutdown. It is important to note that how a behaviour terminates is important. A behaviour's return value reflects on the behaviour selection. If an action terminates with a FAIL code, for whatever reason (the most likely being that the relevant object has disappeared), that behaviour is summarily terminated and its relevance is re-computed from action block 0, as it is assumed that action block n always relies upon successful completion of action block $n-1$. This also prevents a failed action from being continually re-called by accident.

Notice that action blocks are implemented in sequential order.

3.4.4 Behaviour Execution

Behaviours are called from a separate behaviour execution process. This parallelism enables the active behaviour to be continually updated without waiting

for the current behaviour and action functions to return, thus behaviour execution is non-blocking for the rest of The Architecture.

This process is simply a continuous loop that calls the current active behaviour function, which is blocking to this process. Aside from termination flag handling, there is no other functionality in this process.

3.4.5 Behaviour Evolution and Historical Notes

The behaviour selection processes described above is the culmination of a lengthy and rather painful evolution. Earlier versions had a murkier division between behaviours and actions (the original design had no separation at all). Behaviour selection long remained an ad-hoc and subjective process with much vaguer definitions and an internal rising imperative "stimulus" instead of relevance, closer to that discussed in Section 2.4.

The relevance based system was considered cleaner and less arbitrary – the original stimulus design struggled to reconcile a disconnected internal imperative with the object existence and employed an arbitrary function to do so. With the benefit of hindsight a better system that represents a potential advance on relevance is discussed in Section 5.1.

3.4.6 Survival Selection and Execution

The final process relates to the preservation of the individual agent. At the time of writing only "avoid" was supported as a survival action.

Survival selection is a hard-coded process. Unlike behaviour selection, which relies on the insect-derived threshold model and relevance, survival selection is a specific and unalterable procedure for each survival action. This reflects the fundamental expectation that there will be very few survival actions necessary and that they will all be self-evident. Agent survival is the exclusive province of the designer. Users are not empowered to alter or control the survival processes of their agents, because survival is individually focused and does not necessarily act to advance the common goals of the swarm.

In this implementation of The Architecture, avoidance selection was fairly simple. The object list was scanned on a continual basis to see if any object fell within a minimum range (0.25m for the real-world robots and twice the robots' radius, 0.17m, for the simulated).

If an object fell within this physical range and its type was not set as an exception by the active behaviour, then a random number (between 0 and 1) was produced. If this random number was less than or equal to the avoidance object's confidence, then the `motor_override` flag was set in shared memory. This flag was the signal for the survival execution process to call the avoidance action.

The random number call acted as a low-pass filter, making it improbable that very brief duration false positive objects would be identified as collision hazards. This filtering was found to be a practical real-world necessity.

In early implementations, the survival execution process called an avoidance function that was sensitive to the `motor_override` flag, returning if this was re-set. Later implementations used a blocking avoidance function that executed a sequence of time-controlled motor revolutions without reference to the outside world once commenced (see Section 3.3.1 for more on blocking actions).

3.5 Designing Behaviours

When designing new behaviours, the swarm user must follow a deductive process, sequentially reducing large-scale abstracts to the atomic units of the architecture. They must keep in mind the limitations of the individual swarm agents and the guiding principles of the swarm itself to avoid making overly complex behaviours or inappropriate demands on swarm agents.

In Section 2.2 we saw how no swarm agent has the wherewithal to grasp the entirety of any project, their entire universe is circumscribed by what is before them – the object list. What they can do is limited to the action functions. These are ultimately the only tools the user has with which to implement their project.

Recall from Section 3.4 that a behaviour is a sequence of actions representing the smallest sub-component of a task that may be attempted in isolation by a single agent. Thus behaviours are designed by considering the action sequence necessary to achieve this single goal; which actions are needed and in what order? What are the relevant objects for each action block? In what logical relationship do they exist? What is the active object type needed by each action?

By considering these questions behaviours are constructed, action block by action block. For this project these stages were hard-coded in both the real-world robots and the simulation – a higher level interface would simply have added to the complexity of the code, without having any effect on architecture execution. More user-friendly possibilities are discussed in Chapter 5.

Thus the design process can be summarised as follows:

1. Decide on the desired outcome of the project. eg: the construction of a structure, such as a insect nest. This is a task.

2. Determine the smallest components of these tasks that may be done in isolation, which objects need to be gathered, which need to be grouped or moved, etc. These components are behaviours. For example an "assemble component rooms" task implies that an "acquire building block object" behaviour exists.

3. For each behaviour determine the action blocks, the relevant objects and their logical relationship necessary for relevance. For instance: an "acquire building block object" behaviour would need to employ the "move to object" and "grasp object" actions. An instance of the "building block object" would need to be present for this action to be relevant, and so on.

See Pirjanian (2000) for another example of a related behaviour design process.

The designer needs to bare in mind that each behaviour is only one goal. Complexity is achieved through the interaction of many behaviours and robots, *subsuming* lower level behaviours into more complex outcomes[10]. This is related to Brooks's (1999) original subsumption approach.

[10]While it is in theory possible to define only one long, complex behaviour that does "everything", this is strongly discouraged. Experience shows that a behaviour of more than three action-blocks is unlikely to be completely executed. The odds of all criteria being filled in the correct order (with faulty real-world execution and sensing) is simply too great.

It should be emphasised that only step 3 requires any actual computer input, the proceeding steps are simply intended to illustrate the deductive process which is necessary for behaviour development.

The final point that needs to be understood relates to logical order. It is possible that behaviours would need to be executed in a logical sequence. However, there are no features in The Architecture that permit this to be explicitly stated. The designer cannot specify that behaviour X must be executed before behaviour Y becomes relevant. Instead, this is implied through the objects defined as relevant in each behaviour. In accordance with stigmergic principles, robots react to the results of previous robots' actions. In this way temporal precedence is enforced, not by a single robot's actions, but by the actions of the swarm as a whole.

3.6 Summary – The Complete Robotic Insect

Thus the full system is an architecture in five processes[11]. Although it is possible to accumulate all these parallel processes into a single sequential process (as was done with the simulation), it is easier to consider them to be fully parallel, running independently and communicating through a central, shared repository – as depicted in Figure 3.1.

The five Architecture processes are:

Sensor Process

The only input for data from the outside world. All sensor data is processed and analysed in this process only, where it is turned into objects in the list (see Section 3.2). Filling the Object List is the only responsibility of this process. As part of that responsibility, the zeroth object (see Section 3.4.2) is maintained here.

Behaviour Selection Process

The behaviour selection process is responsible for updating behaviour relevance (by using the relevance calculation described in Section 3.4.1 and illustrated in Algorithm 1), behaviour thresholds and nominating the active behaviour by setting the `active_b` flag. If a behaviour has been active for too long, the null behaviour will be forcibly engaged for a set period, preventing failed behaviours from being continuously run.

Behaviour Execution Process

Responsible for calling the behaviour functions, which in turn call action functions. If a behaviour returns a fail code, the null behaviour and tasks are set as active. Actions and behaviour functions are sensitive to the `active_b` flag and will terminate if this flag changes (except if blocking). If the `active_t` flag changes, the active behaviour will change as a matter of course.

[11] If the reader peruses the code for the live robots they will find that there are actually six separate processes and a final "main" The sixth process is a data-logging process and is supernumerary. The final main return is largely historic and handles motor communication, which, for technical reasons, is easier to do as a separate process to the action functions. Note also, that the processes could also be threads.

Survival Selection Process

Assess the necessity for the survival actions. In the current version of The Architecture only "avoid" is supported here. If the survival action is needed, set the `motor_override` flag in shared memory.

Survival Execution Process

Calls the relevant survival actions (if any).

The complete Architecture may also be seen in Figure 3.1. See Appendix B for implementation.

Chapter 4

The Swarm Architecture: Testing

"...the Answer, the Ultimate Answer, to Life the Universe and Everything...
is...
is...
42!"

"...we are going to get lynched, do you know that?"

— Douglas Adams,
The Hitchhiker's Guide to the Galaxy

Summary

This section discusses The Architecture in practice. In this project implementation is intended as a proof-of-concept, and employs idealised insect-like swam tasks to demonstrate The Architecture's functionality. The Architecture is implemented upon two distinct insect-like swarm agent designs (real-world and simulated), as well as three different tasks.

Contents

When implemented, The Architecture produces swarm-like cooperation. The robots function adequately as both individuals and a collective.

The Architecture demonstrates:

- Successful execution of tasks set.
- Adaption to new swarm-tasks without fundamental modification.
- Successful implementation on two distinct swarm robot designs – one real-world, one simulated. For details on these robots see Appendix B.
- Scalability, tasks with 100 agents and four agents were attempted with comparable results.
- Emergence of cooperative aspects – such as a rough division of labour and swarm-wide agreements – without the user explicitly requesting these aspects.

We conclude that The Architecture is functional in implementation and workable as a concept.

In practice, it was found that successful implementation of The Architecture's current form is heavily dependant on:

- Comprehensive local perception. As well as clear identification of object type, robots must be able to discriminate between atomic objects and semi-complete structures.
- Comprehensive searching.
- Sufficient robot density. Due to lack of communication, information does not propagate through the swarm. The probability of at least one robot encountering all relevant objects must be high.
- Environmental containment. Lacking a swarm cohesive mechanism, robots will disperse without firm bounds on their universe. Peer-to-peer communication may address this – see Chapter 5, Future Work.

4.1 The Tasks

The Architecture was tested upon three tasks. These tasks were designed to be idealised versions of swarm tasks, reduced in complexity to make implementation tractable with limited time and physical resources. They were intended as a test of The Architecture's functionality, not an exhaustive test of every possible swarm task.

The three tasks were implemented in both simulation and hardware. Simulated results were the most successful (as real-world problems did not intrude) and provide the basis of the observations. Real-world tests were intended to demonstrate the efficacy of the simulation.

Ball-Passing

One target object (a "ball") was passed between the swarm members, the task objective being to keep the target object in motion.

This simple task was the fundamental task used for initial testing, debugging and the gathering of basic quantitative data in both the simulation and real-world experiments. The length of time each robot gripped the ball was measured and used as a basis of comparison between swarms. A broadly equal division of labour was found, without explicitly programming for this result.

Additionally, such object transfer is redolent of Turner's (2011) "bucket brigade" termite building (Dorigo et al. 2000). The individual agents in a swarm may be too limited to move an object from A to B without assistance. The distance may be too far, the agent too weak, or its perception too limited to the location of grasp point B in any more than a general directional sense. Thus it is logical for one agent to pass its burden to another if it can find no more constructive use for it in its immediate environment.

Grouping1

Gathering n scattered objects and combining them in a group. The group could be in any arbitrary location in the arena.

By executing this task, the swarm demonstrated its ability to explore the environment, manipulate objects, assemble them in some more ordered structure and maintain that structure over time. Such gatherings are the basis of foraging, construction and other practical tasks.

This task demonstrated the auto-catalytic self-organisation discussed in Dorigo et al. (2000) and Garnier et al. (2007). One robot combining two objects creating the seed of the group. All agents subsequently used this start, placing their objects at the same location without global knowledge.

To prevent the groups from be disassembled, this task introduced a new object type – the "group" object (see Section 4.3.4 for a discussion on the necessity for this). In the real-world this was any collection of target objects greater than one within 0.5 metres of each-other (later dropped to 0.35 metres, from experiment 4). Although the depth detection in the real-world robots had a marked tendency to under-read. In the simulation, objects needed to be within one robot diameter (17cm) to qualify.

The component objects in the group maintained their individual existence in the object-list as target objects.

Simulated experiments are run over a 3600 "second" period, but have usually achieved a group within the first few hundred seconds, thus real-world experiments are of (roughly) 1200 seconds.

Grouping2

Identical to Grouping1, except that the objects were required to be clustered around a specific point – a central pillar – marked by a third object. This task indicated that the user may exert some measure of control over the self-organisation process, by stipulating an arbitrary collection point.

Both grouping tasks were also successful in a very large-scale (800 target objects, 100 robots) simulation to demonstrate the scalability of The Architecture.

4.2 Ball Passing Task

4.2.1 Task Behaviours

To implement the ball passing task, both the real-world and simulated robots were equipped with the following three behaviours:

ACQUIRE – move to and grasp the ball target object.

action block one – Move to object

Relevant Objects:	target-obj = EXIST
	captured-obj = NOT_EXIST
	capture-by-other = NOT_EXIST
Exception Objects:	target-object
	captured-obj
Action:	**move-to-object**(target-obj)

action block two – Grasp object

Relevance relations and Exception objects are unchanged.
Action: **grasp-obj(** target-obj **)**

BALL_TO_AGENT – pass the captured ball.

action block one – Move to agent

Relevant Objects:	captured-obj = EXIST
	agent = EXIST
Exception Objects:	captured-obj
	agent
	captured-by-agent
Action:	**move-to-object(** agent **)**

action block two – Drop ball

Relevance relations and Exceptions are unchanged.
Action: **drop-object()**

RECEIVE – if an agent with a ball is sighted, move to receive the ball. This assumes that the other agent wants to pass.

action block one – Move to agent.

Relevant Objects:	captured-by-agent = EXIST
	captured-obj = NOT_EXIST
Exception Objects:	none.
Action:	**move-to-obj(** capture-by-agent **)**

There are no other action blocks in this behaviour.

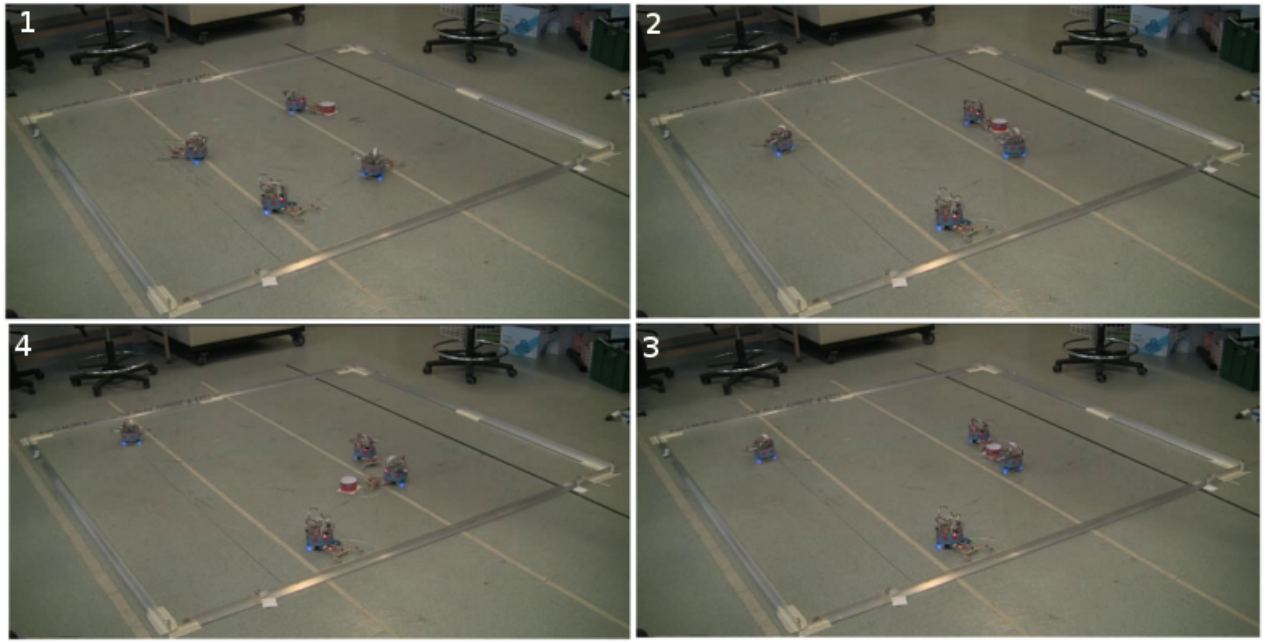

Figure 4.1. A sequence of stills from 3:59:012, passing experiment 13, showing an instance of ball passing. Clockwise from top left, the first robot with the ball, approaching another robot, transfer (in this case resulting in collision), the second robot with the ball.

4.2.2 Simulation Results

For thirty six-hundred second simulations, each robot held the ball for an average of approximately 9.215 percent of the simulation ($\sigma = 4.377\%$).

4.2.3 Real-World Results

Table 4.1 shows the mean time an individual robot held the target ball as a percentage of the overall task. This is broken into four categories: the individual mean for scoop-equipped robots, the individual mean for hook-equipped robots and the means for both as computed from the total experimental duration as measured from a wall clock[1].

Table 4.2 gives the overall time that the swarm had the ball in motion as a percentage of the overall experiment duration. This data is broken down in the same fashion.

It should be noted that the real-world times are not exact (see Appendix B). They are intended to represent reasonable, although rough values that give a basis for demonstrating The Architecture as a concept.

Experiment	**Mean Time Gripped (% – individual)**	**standard deviation**
scoop robots, wall-time mean	7.10	2.19
scoop robots, internal time mean	6.11	1.83
hook robots, wall-time mean	11.45	2.36
hook robots, internal time mean	10.44	2.11

Table 4.1. Individual real-world robot results for the ball-passing task, this is the time that individual robots spend gripping the "ball" as a percentage of the experiment. These results are an average of all robots over all experiment. Separate results exist for the scoop robots and the hook equipped robots and both sets are computed based on the wall-time length of the experiment and the robot's internal measurement. The time each robot spends with the ball in its scoop is always its own measurement.

4.2.4 Observations and Discussion

Conclusions From the Data

From Table 4.1 and 4.2 we can see that the results of both real-world sets are roughly comparable to the simulated results.

In all cases, the robots hold the ball for a roughly equal percentage of the experiment (as evidenced by the fairly low standard deviation) and the aggregate

[1]The stopwatch used as the "wall-clock" had a precision of one-hundredth of the second. The precision of the results reflects this. see Appendix B for details on time.

Experiment	Mean Time Gripped (% – swarm)	standard deviation
scoop robots, wall-time mean	28.38	2.97
scoop robots, internal time mean	24.45	2.84
hook robots, wall-time mean	45.81	2.84
hook robots, internal time mean	41.76	5.39

Table 4.2. Swarm real-world results for the ball-passing task. This is an average of the summed times for each experiment and is separated into scoop and hooked robots and clock and internal time as previously.

time the ball was in motion was a reasonably significant proportion of the total time.

From this we conclude that The Architecture is:

- Successfully completing this task in these environments on these robot platforms, with the platforms comparing reasonably well.
- Capable of useful emergent behaviour. The robots all hold the ball for comparable times, yet this rough division of labour was not explicitly encoded into the behaviours.
- Capable of keeping a swarm of four robots engaged in a task in such a way as the principle objective of the task is being fulfilled for a significant percentage of the time.

Comment from Observations

Several general observations can be made from the experiments and debugging experience:

- The "receive" behaviour had a noticeable effect on the robots. By using the captured-by-other object type, this behaviour enabled a robot's overall behaviour to be affected by the actions of another agent. On several occasions one or more robots were observed following the ball-carrying robot as it searched for a passing mate. This is a clear instance of recruitment, as one machine attracts others to it by the execution of its task.

  Frequently, these recruitment episodes manifested themselves as a "staring contest," with the ball held static between two facing agents[2]. We believe that in these situations, both machines identified the ball as the property

[2]See also Lerman & Galstyan (2002) for more on how robots may interfere with each-other. They note that performance growth is sub-linear to group size growth. Further discussion on robot-environment interaction and behaviour-based run-time improvement can be found in Goldberg (2001).

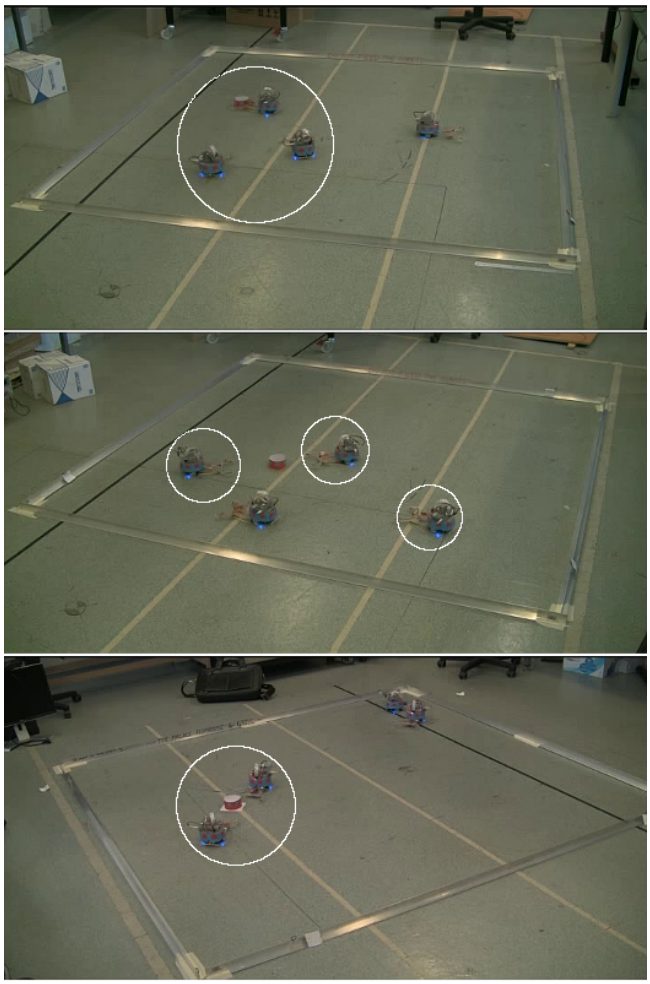

Figure 4.2. Three illustrations of the receive behaviour. From Top: two agents following a third, two agents facing each-other with the ball in between them with a third watching, two agents with the ball between them. From experiment 3, 100 seconds, experiment 1, 154 seconds and experiment 16, 135 seconds respectively.

of the other and, having deposited it, were unwilling to leave. See Figure 4.2 for examples of both types of receive behaviour

This is an illustration of the actions of one agent stigmergically effecting the actions of another.

- Without boundaries on the arena, the robots' search behaviour gradually pulls the swarm apart. A minimum number of robots is needed in an area to allow swarm cooperation.
- Thresholds were noticeably less significant than relevance. Robots did not develop noticeable specialisation – one machine always grasping the ball. Each behaviour's relevance criteria usually provided a logically exclusive set of conditions.
- A series of bugs and hardware failures, especially with the real-world robots, is discussed in Appendix B. With the knowledge gained as part of the development discussed in this appendix, more effective robots could be built. However, this project is a proof-of-concept, which has been demonstrated. There was insufficient time and resources to justify rebuild the real-world robots from scratch.

4.3 Object Grouping1

4.3.1 Task Behaviours

ACQUIRE – move to and grasp the ball target object. Identical to its counterpart in the ball-passing task, except that target-obj-group = NOT_EXIST.

BALL_TO_AGENT – pass the captured target object. This behaviour was identical to its counterpart in the ball-passing task, except that target-obj-group = NOT_EXIST. This behaviour was eventually discontinued as a distraction (experiment 7).

RECEIVE – if an agent with a ball is sighted, move to receive the ball. This assumes that the other agent wants to pass. This behaviour was identical to its counterpart in the ball-passing task.

GROUP1 – move captured target object to another target object or existing group of target objects.

action block one – Move to object

Relevant Objects:	captured-obj = EXIST
	target-object = EXIST
Exception Objects:	captured-obj
	target-obj
	target-obj-group
Action:	**move_to_obj(** target-obj **)**

action block two – drop object

Relevance relations and Exception objects are unchanged.
Action: **drop_object()**

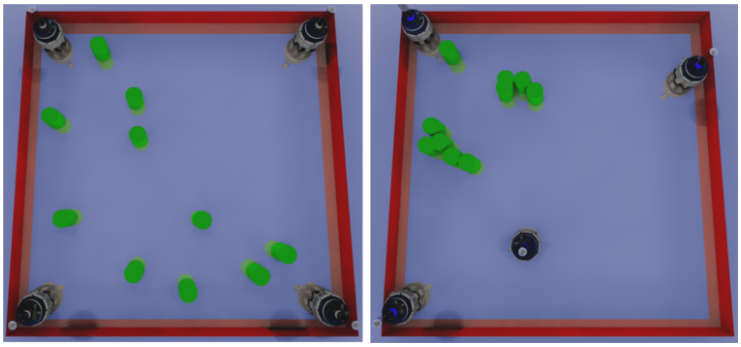

Figure 4.3. The object Grouping1 task in the simulator. Left: at the beginning of the simulation with the robots in the corners and the targets randomly distributed. Right: the final stage after 20 simulated minutes, two clear groups and one spare.

It should be noted that for the real-world robots, a fully blocking drop action was employed (from experiment 1) and complemented from experiment 2 with fully blocking avoidance. This was in an attempt to overcome the perceived poorer execution in the real-world robots. All of these modifications led to incremental observed improvements in individuals.

4.3.2 Simulation Results

Reasonably successful grouping was achieved by the simulation – see Figures 4.3 to 4.10.

The Grouping1 task was quantified by investigating the nearest neighbour distance for each target object. The assumption was that as the simulation progressed, the mean nearest neighbour distance would decrease (as would the variance). Eventually, it should converge on the "grouping distance" (two or more objects within "grouping distance" are seen as a group by the robots). Figure 4.4 shows that this this mean distance is usually below the grouping distance for a all simulations, with Figure 4.5 showing the complete evolution of the mean nearest-neighbour distance for a single simulation selected at random.

These results show that a minimum group size is reached, but is not necessarily stable. The group may expand and contract several times over the course of the simulation.

The results of the large-scale simulation (800 target objects, 100 robots, 20×20 metre arena) can be seen in Figures 4.6, 4.7 and 4.8. Over the course of the 30 minute simulation, the random distribution gave way to distinct clusters of target objects.

It is noted that the clusters of the large-scale simulation are not of equal size, as can be seen in Figure 4.3. Collections of target objects larger than one are considered "groups" by the robots and are generally left alone.

Figure 4.4. Percentage of each simulation that target objects within a group for the small-scale Grouping1 simulation. This is computed by measuring the mean nearest-neighbour distance for each target object. Each simulation ran for 3600 seconds, contained four robots and ten target objects.

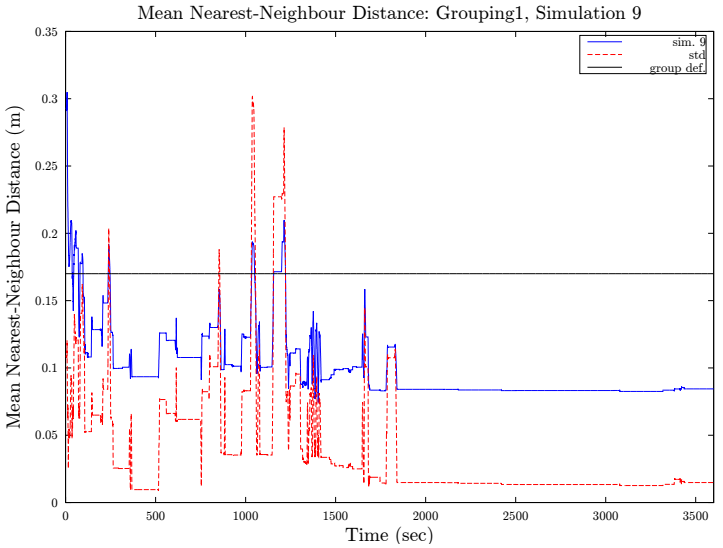

Figure 4.5. Plot of the target-object nearest-neighbour distances for a typical Grouping1 simulation run (chosen at random). This plot shows the mean distance of one target object to another over the experiment.

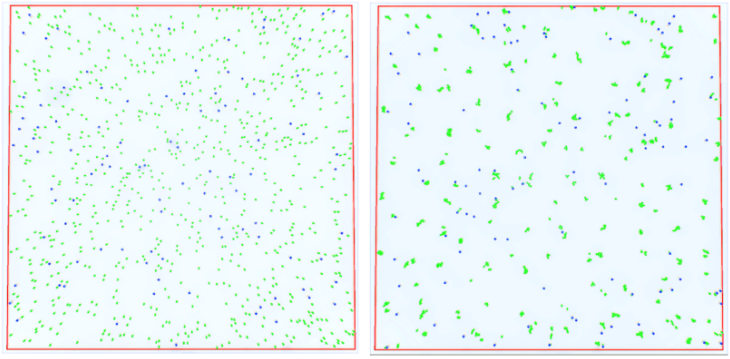

Figure 4.6. The object grouping task in a very large (20 by 20 metre) arena. The objects were initially arranged in a uniform random distribution (left), while small clusters of target objects are discernible at the end of the 30 minute simulation (right). In this simulation there were 800 target objects and 100 robots.

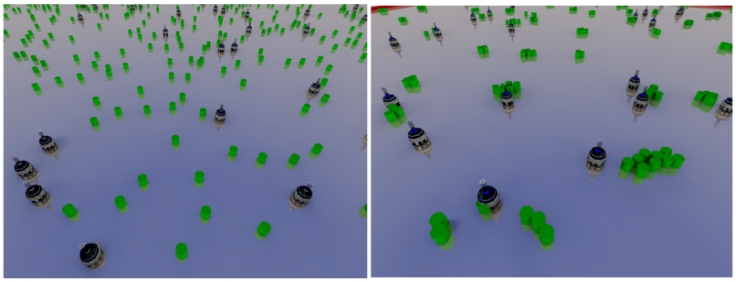

Figure 4.7. A close up view of the large-scale Grouping1 task, left: the initial uniform random distribution (slightly skewed by the image perspective), right: the grouped objects at the end of the 30 minute simulation.

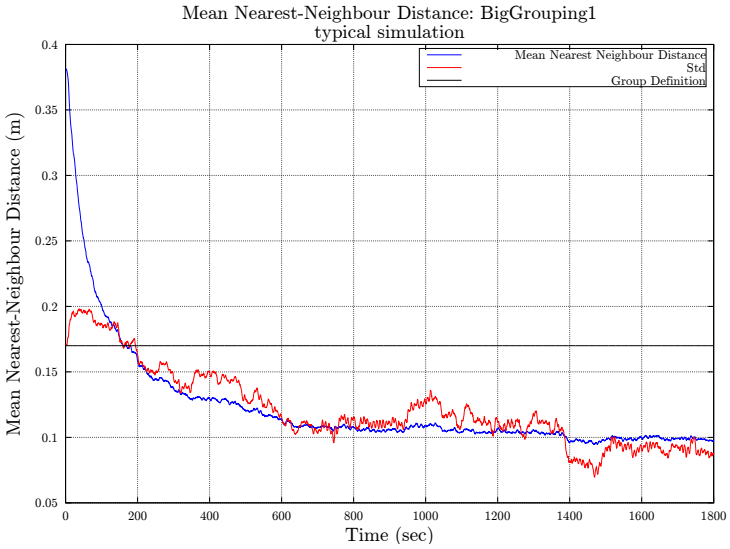

Figure 4.8. The mean nearest neighbour distance for objects in the very large arena, Grouping1. The mean distance trails off as the target objects are grouped closer and closer together. This is a typical simulation.

4.3.3 Real-World Results

The real-world results are qualitative for this experiment. In the absence of the simulator's easy ability to record the position of objects[3] we shall discuss observations from the experiments.

The real-world results of this task depended critically on the availability of the gripper. When executed by gripper equipped robots small groups could be observed, although not with the same speed, clarity or stability that had been observed in the simulation[4].

The gripper tests are ambiguous as the target objects still had to be pushed from the walls. This was because the protruding forward touch sensors would be triggered before the gripper had a chance to engage. The external movement of the target objects certainly adds to the "grouping" cohesion. However, occasionally, groups were observed in clear space (see Experiment 2 from 215 seconds and Experiment 3 from 55 seconds) and the instances of individual agents grouping objects are too clear to be overlooked, although their groups are usually short-lived.

Without the gripper the result was decidedly negative. Although groups did form, they were clearly an artefact of the arena/robot interaction – as robots touched the wall they backed up and turned according to the avoidance action discussed in Section 3.3.2, which caused the scoop to lose control of the target object. Thus target objects had a higher probability of being dropped at the walls and in corners than anywhere else, forming loosely defined artificial

[3] Over-head cameras and tracking could have been employed, but the constraints of time and resources rendered this impractical.

[4] See Appendix B for more on robot gripper development.

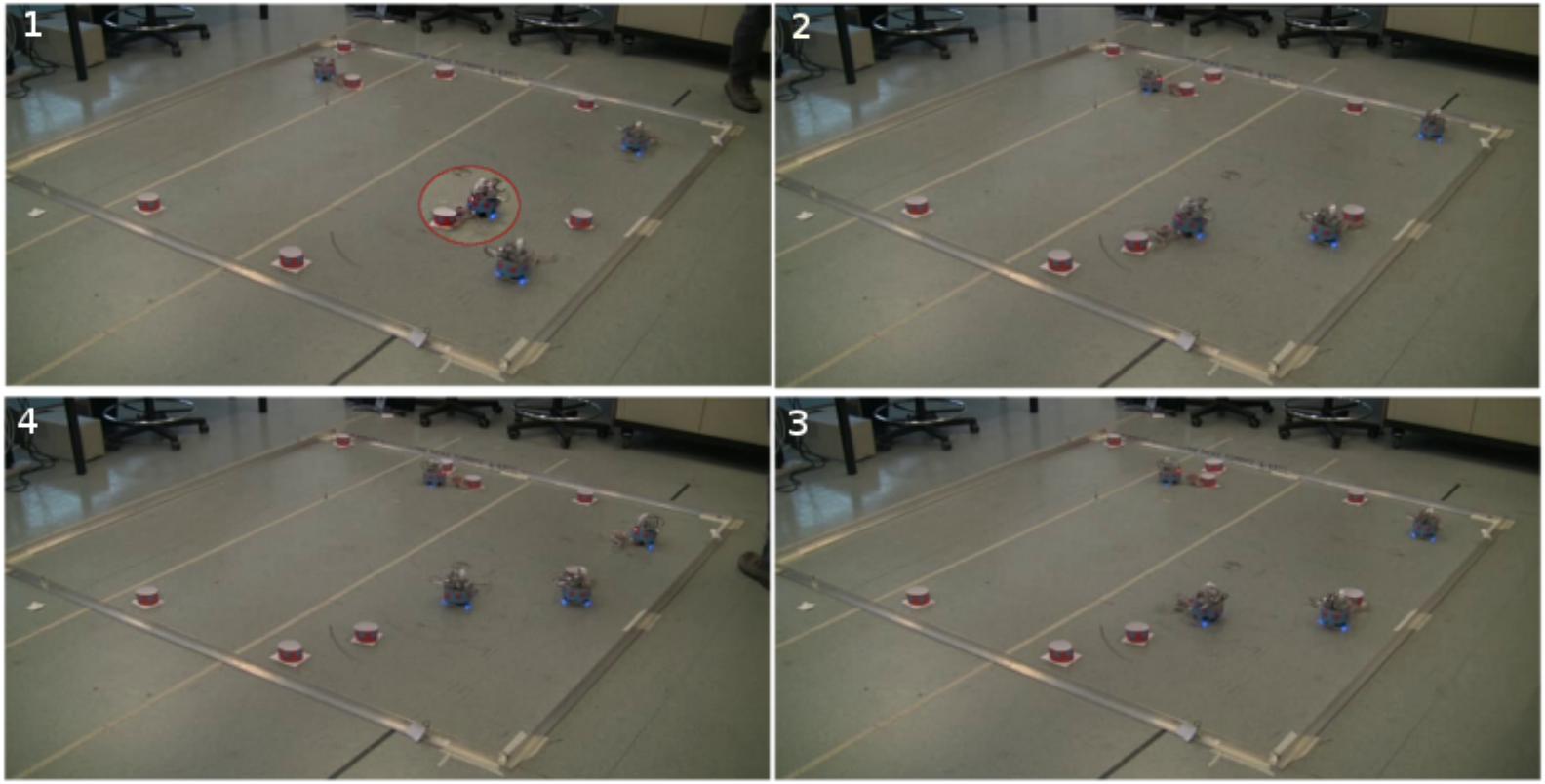

Figure 4.9. Two target objects being grouped in the Grouping1 task. Clockwise from top left: a robot (circled) with a captured object bringing it towards a Target object, the captured object being dropped and the robot backing up, the robot turning away from the new group, the robot resuming searching. From Grouping1, experiment 7, 42.5 seconds.

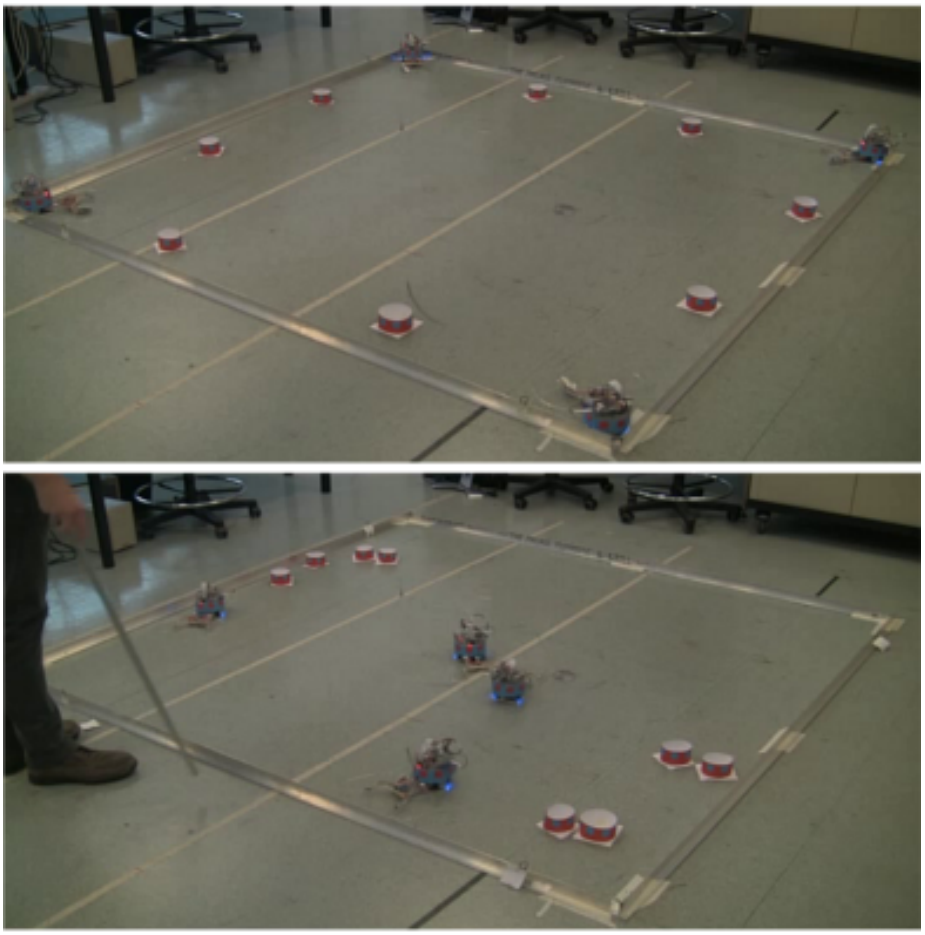

Figure 4.10. Grouping1 with grippers. Top: the beginning of the Grouping1 task showing the dispersal of the target objects. Bottom: a later stage in the same experiment showing the objects in two clusters. From experiment 6, 3.5 seconds and 220.2 seconds, respectively.

"groups". At no time were stable groups of target objects observed to form in free space. Although this illustrates how environments can be exploited to achieve swarm objectives, it does not demonstrate The Architecture *per se*.

4.3.4 Observations and Discussion

Overall, the Grouping1 task illustrates the amplification of initial random fluctuations necessary for self-organising cooperation (Garnier et al. 2007). The objects begin in a random initial configuration, one robot makes a group of two objects. The other robots in the swarm all place their captive objects near this initial seed pair. There is no explicit communication of the first pair's location, yet a global "decision" is made regarding the final group's location.

Simulation

The simulated version of the Grouping1 task produced clear groups, occasionally more than one per simulation. Groups were stable in the long-run, but prone to periodic disturbances as outlying target objects were "snipped off" by passing robots. These snipped objects were usually returned to the main group eventually – as can be seen in Figures 4.4 and 4.5 the objects are grouped for the majority of the experiment.

It is believed that these instabilities stem from group description and shape. Groups are built in an unstructured fashion and often grow "tails" as objects are placed next to the first target object an agents sees. These tails are vulnerable to wandering agents who can see the lone object, but not the rest of the group.

When a stable group size is eventually reached, its nearest neighbour distance is generally much smaller than the definition of a group. This is probably due to objects being "bumped" by robots as they wander the arena. As there is no real resistance for the robots that side-swipe an object, the object is shunted closer to its group-mates. If the object is moved out from the group it is usually snapped up by another robot as a free object and placed back in the group. These factors would tend to condense the groups as far as physically possible.

Figure 4.3 shows the initial random distribution reduced to two distinct groups. Multiple groups occasionally form because the robots cannot perform quantitative analysis – a group of two target-objects is a target-object-group, as is a group of n objects. The object type classification does not include a rating system or means of assessing the membership degree of types in a fuzzy sense. See Section 5.1.3 for more on this.

The large-scale simulations reveal that The Architecture functions just as well with very large swarms as it does with the very small. Figures 4.6, 4.7 and 4.8 illustrate that, just as in the small-scale version, coherent groups consistently form and, as Figure 4.8 shows, maintain great stability over time.

The large-scale Grouping1 task illustrates an important component of swarm behaviour; the ability to cooperate on tasks on a much greater scale than the agents are individually capable of.

The robots do not have enough memory or sensor range to detect all objects in the environment. Despite this, grouping follows much the same pattern everywhere. This emphasises that swarm agents do not need global maps, plans or communication to create a global structure from local interactions.

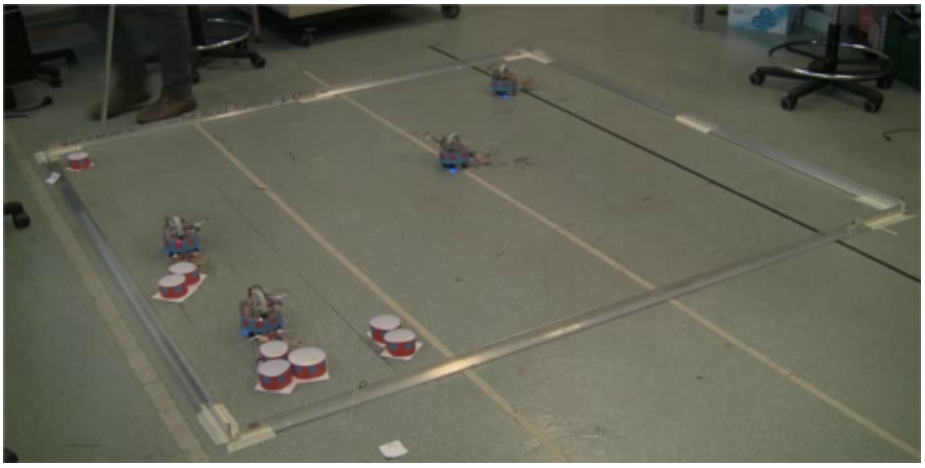

Figure 4.11. Illustration of the problems of target-object occlusion in the Grouping1 task. The two left-hand robots are trying to place their captured object near other target objects, unaware that they are pushing such objects before them. From experiment 2, 357 seconds.

Real-World

Grouping was occasionally observed in the real-world task (with hooked robots). However it did not have the same crisp definition seen in the simulation.

This problem was caused by rogue members of the swarm demolishing groups constructed by others. Nascent groups of target objects were observed, but frequently had no longevity. This was observed in the simulator, but was more pronounced in the real-world.

It is believed that this self-destructive behaviour was based in the limitations of the hardware, the robots could not reliably perceive the target object groups. The problem is illustrated in Figure 4.11, where an agent is searching for other target objects to deposit its capture, unaware that there are two more in front of its scoop.

Occlusion was the biggest problem. When two target objects were in close proximity, there is only a finite zone where the object markers on both objects were visible to an observing robot, the two objects occluded each others' identifying markers.

In The Architecture, information promulgates between agents by means of observable changes in the environment, either a change in the physical orientation of objects or a change in object type. To build any type of structure the robots must be able to distinguish between three phases of object: "component", "half-state" and "finished" which must be explicitly or implicitly defined as object types.

A "component" object would be the atomic object of the structure, in this case the single target object that comprises the basis of the groups. Other examples would be bricks for a wall, debris for a midden, etc. A "half-state" object is the incomplete structure, two target objects grouped together, two stacked bricks, etc. Notice that there would not need to be a distinction between

a group of two target objects and a group of N target objects, any number greater than one is acceptable to define a half-state object.

The final type phase is "finished", designating a completed structure. This type would probably not be defined explicitly, as an individual swarm agent would be unlikely to possess the perceptual scope to comprehend the completed structure, unless it was very small. Instead, the finished phase would be defined by a lack of component objects or locations in the half-state structure to place them. See Section 2.5.2 for a previous example of this.

The instability of the real-world grouping task illustrates the importance of these distinctions. Because the individual agents in the swarm do not know explicitly that their companions have completed a structure, the completion must be self-evident if they are not to pull it apart again. For any structure to be completed the swarm agents must be able to distinguish reliably between the atomic components of a structure and the structure itself.

In this case, the robots' limited perception and the identically sized objects ensured that such distinctions could not be reliably made. This also demonstrates that the target component objects should not be large relative to the robots' sensor field of view – insect-like, the object or part of an object must be small enough for an agent to manipulate.

4.4 Object Grouping2

The Grouping1 task was a qualified success, but in that task the swarm was free to assemble groups wherever it saw fit. The groups of target objects form in clear space with no further influencing factor other than the original position of the robots and target objects.

While success in this task is not insignificant, there will be times when the user wishes the swarm to move objects to an arbitrary point. Owing to the object list requirements this point must be an identifiable object, but its location in the arena may be arbitrary. Thus the Grouping2 task was attempted, requiring the swarm to group target objects around a central object or "pillar".

4.4.1 Task Behaviours

ACQUIRE – move to and grasp the ball target object. Identical to its counterpart in the ball-passing task, except that target-obj-group and target-obj2 = NOT_EXIST.

BALL_TO_AGENT – pass the captured target object. This behaviour was identical to its counterpart in the ball-passing task, except that target-obj2 = NOT_EXIST.

This behaviour was included in the belief that – as the robots are very limited and there will occur times when a robot has a captured object and cannot see the central pillar – it is best to give a captured object to a passing robot who may find the pillar rather than do nothing. This worked fairly well in simulation, however it had to be discontinued in later tests of the real-world architecture as it created too much confusion. As the real pillar was much harder for the real robots to find, they tended

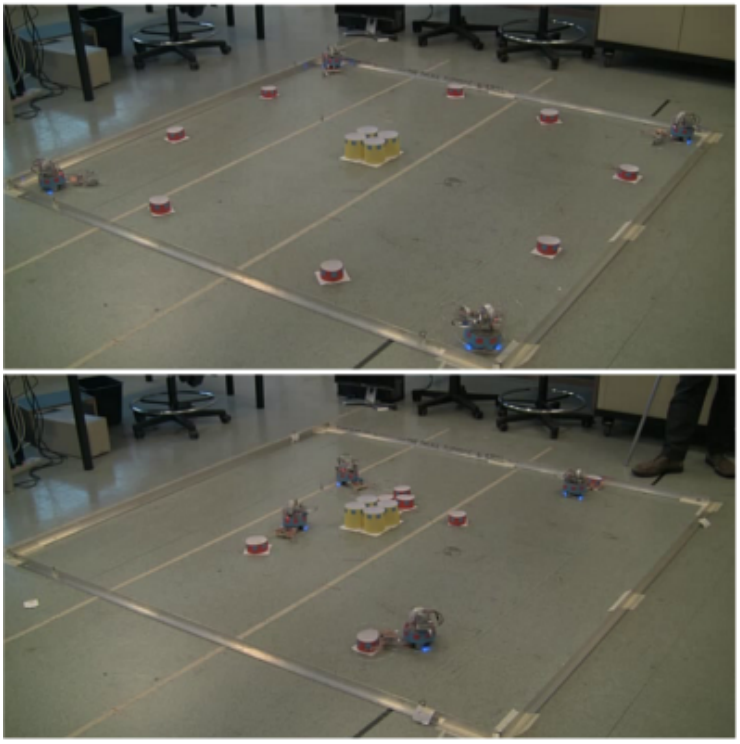

Figure 4.12. Top: the beginning of the real-world Grouping2 task, showing the target objects, the robots and the central "pillar" object in their initial positions, bottom: the same experiment after approximately 624 seconds, showing the partial grouping of the target objects. From experiment 7.

to spend the majority of their time passing target objects to each-other. Some target objects were in constant motion, never finding their group.

RECEIVE – if an agent with a ball is sighted, move to receive the ball. This assumes that the other agent wants to pass. This behaviour was identical to its counterpart in the ball-passing task.

GROUP2 – move captured target object to another target object or existing group of target objects.

action block one – Move to object:

Relevant Object Types:	captured-obj = EXIST target-obj2 = EXIST
Exception Object Types:	captured-obj target-obj2 target-obj1 target-obj1-group proximity-obj
Action:	**move-to-obj(** target-obj2 **)**

action block two – drop object:
Relevance relations and Exception objects are unchanged.
Action: **drop-object()**

For the real-world robots, a fully blocking avoidance function was employed from experiment 1 of this task, from experiment 3 onward it was joined by a partially blocking un-grasp action (backup, blocking but not turning). Two channel U and V space object colour recognition for the pillar objects was introduced from experiment 4 onward.

4.4.2 Simulation Results

In simulation the second grouping task was successful both on a small scale and a large. On the small scale, Figure 4.13 illustrates that the robots were able to group all the target objects around a central pillar and keep them there over a 60 minute simulation (although the group is basically achieved in the first 1000 or so seconds). Figures 4.16 and 4.17 show that this was also successful on a much larger scale.

The small-scale simulations measure the distance from the individual target objects to the central pillar directly and Figure 4.13 records the mean of this distance for all ten target objects in the simulation.

The small-scale simulations had four robots and ten target objects.

The large scale simulation results are presented in the form of nearest neighbour distance as in the previous Grouping1 task. This is because, with many pillars, it is impossible to tell which target object belongs to which pillar[5].

[5] "distance to closest pillar" could have been used, but nearest-neighbour was already implemented. This plot is really a quantitative representation of Figures 4.16 and 4.17.

Figure 4.13. Percentage of each simulation that target objects within a group for the small-scale Grouping2 simulation. This is computed by measuring the mean distance from the central pillar for each target object. Each simulation ran for 3600 seconds, contained four robots and ten target objects.

The large simulations were conducted in a 20 by 20 metre arena, with 100 pillars arranged in a two by two metre grid. Grouping was performed with 800 target objects by 100 agents.

4.4.3 Real-World Results

In the early real-world experiments, the scoop robots were unable to group the target objects at the "pillar". The results were similar to those of the scoop robots in the first grouping task, with an inability to grip causing an endless movement of target objects through the arena. Also, most of the target objects were eventually abandoned by the robots at the edges, rather than near the target object. This failure was probably the result of poor object detection (the scoop robots used an early target-object2 print that was the wrong colour).

Later experiments with hooked robots and a different coloured pillar object were more successful (see Figure 4.19), although never quite at the levels of the simulation. The most number of objects grouped at the central pillars was five of eight and this group never attained stability. The real-world robots had significant trouble locating the central pillar, even in its final colour and it is suspected that three colours was the limit of the recognition system. However, clear instances of grouping were observed.

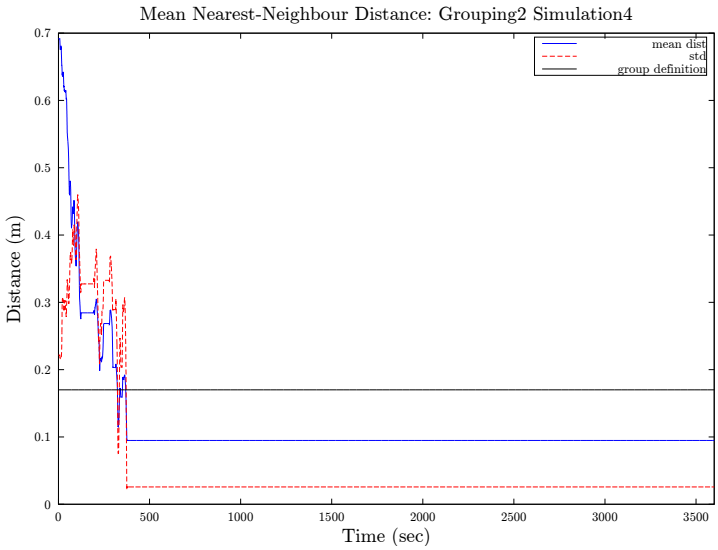

Figure 4.14. One Grouping2 simulation chosen at random, showing the mean distance of the target objects from the central pillar over the course of the simulation.

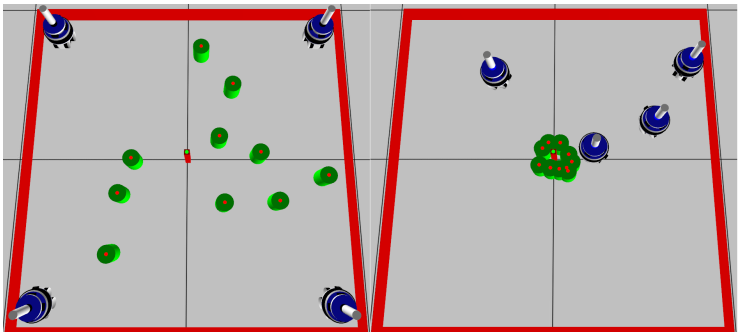

Figure 4.15. Left: the beginning of a typical Grouping2 small-scale simulation. Right: the final grouping result of a typical simulation. The red grouping pillar object is at the centre of the arena.

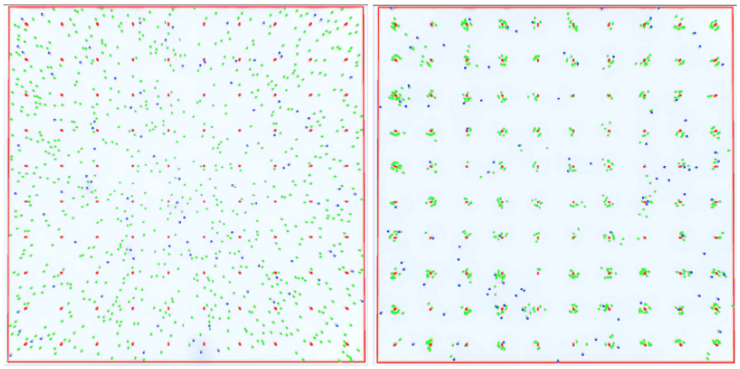

Figure 4.16. The second object grouping task in a very large (20 by 20 metre) arena. The objects and robots are initially arranged in a uniform random distribution with 100 grouping pillars in a two metre by two metre grid (left). Small clusters of target objects are discernible around each pillar at the end of the 30 minute simulation (right). In this simulation there were 800 target objects and 100 robots.

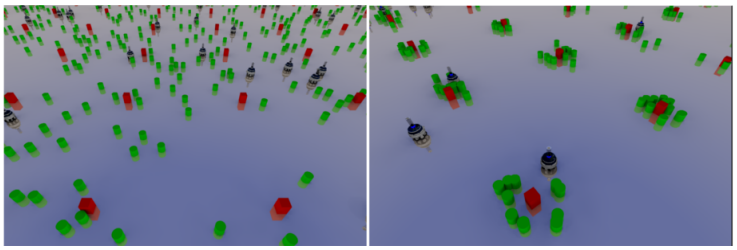

Figure 4.17. A close up view of the large-scale Grouping2 task, left: the initial uniform random distribution (slightly skewed by the image perspective), right: the objects grouped around the pillar grid at the end of the 30 minute simulation.

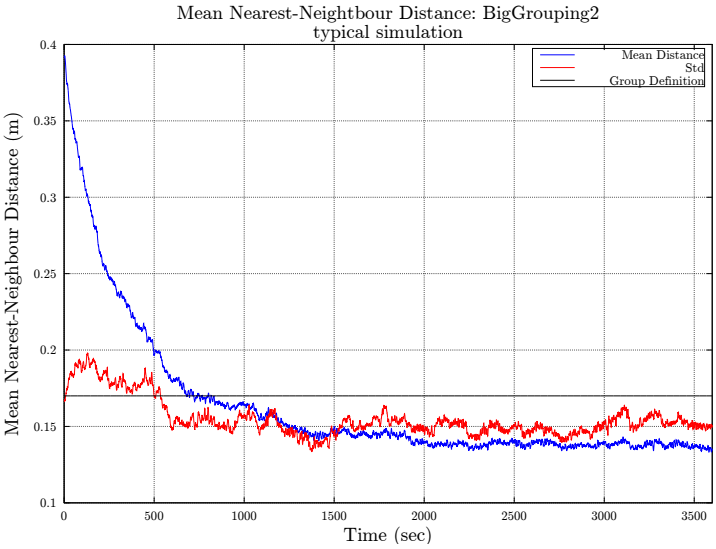

Figure 4.18. The mean nearest neighbour distance for objects in the very large arena, Grouping2. The mean distance trails off as the target objects are grouped closer and closer together. This is a typical simulation.

4.4.4 Observations and Discussion

Simulation

The small-scale simulation achieved notable success in grouping the target objects. Figure 4.13 shows that objects rapidly condense to a small, central group and remain there consistently. There are only a few occasions when a group is disassembled.

This improved group stability is believed to be due to two factors:

1. The measuring process is subtly different in the Grouping2 task. The presence of the central pillar enables us to measure the group as mean distances from the simulation origin (which is the centre of the arena), rather than nearest neighbour distances. Thus for each group we can accurately measure the size of the group.

2. A "better" group definition exists in Grouping2. The Grouping2 robots do not look for existing groups, instead focusing upon the central pillar. Unlike the Grouping1 robots, identification of groups is not critical, the existence or absence of the pillar object is the deciding factor. Any target object near the pillar is implicitly part of the group.

   This means that groups are less likely to form elongated "tails", constituent objects are deposited near a central marker, not the last object dropped.

   Compare the elongated groups in Figure 4.3 (right) with the tight, centralised group in Figure 4.15 (right) to see this.

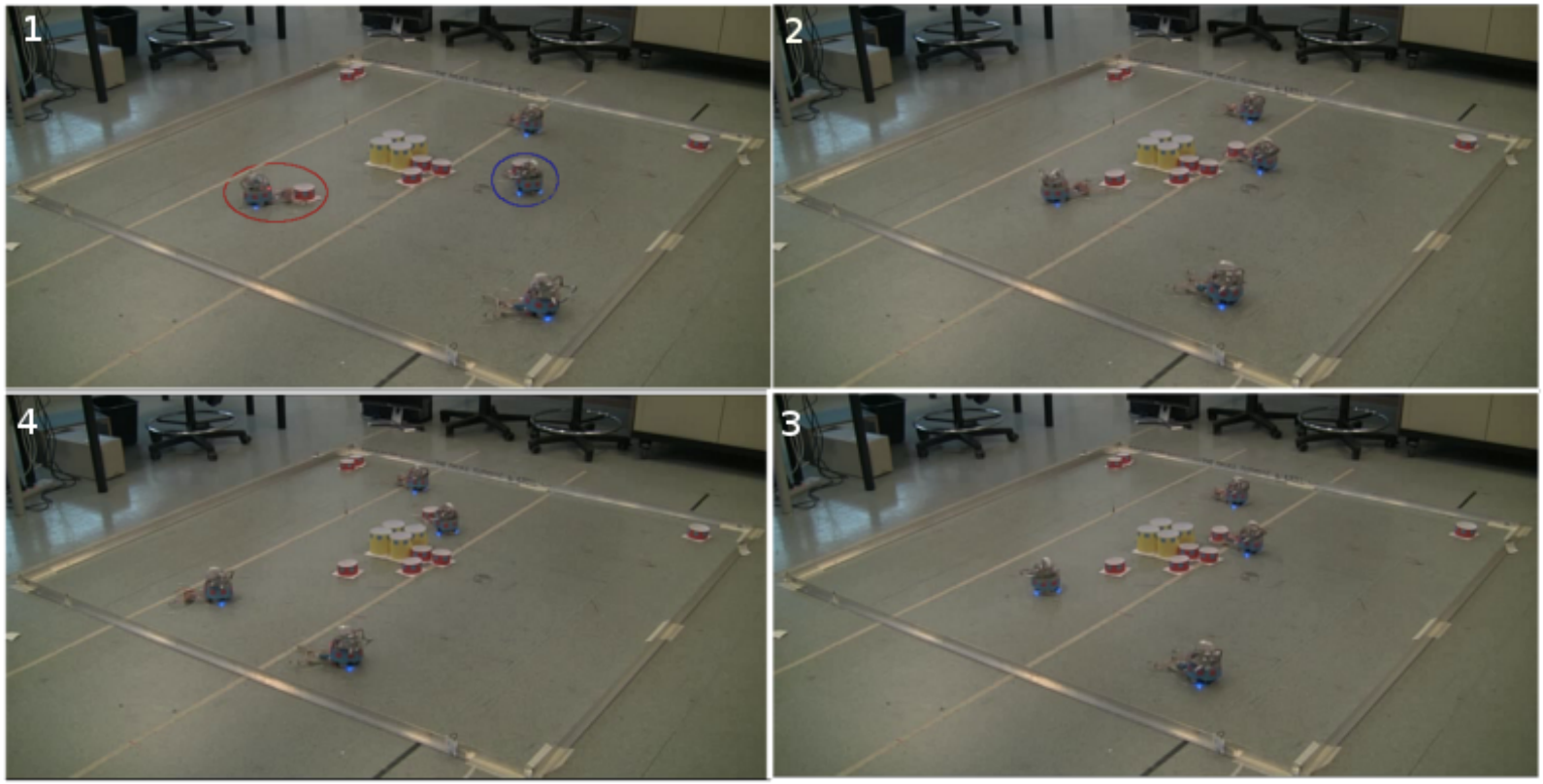

Figure 4.19. Two target objects being grouped in the Grouping2 task (hooked robots). Clockwise from top left: a robot (circled, red) with a captured object bringing it towards the central pillar object, the captured object being dropped and the robot backing up, the robot turning away from the new group, the robot resuming searching. Notice that there are already three objects grouped around the central pillar while another robot (circled, blue) also has captured a target object, but fails to perceive the central pillar. From Grouping2, experiment 4, 557.7 seconds.

Essentially, Grouping2's behaviour set-up is structured to produce more compact stable groups than Grouping1.

The large-scale simulations of both styles of object groupings illustrate the importance of a dynamic environment and agent interaction. Although the robots do not need the direct help of their fellows to move the objects, they do rely on other mobile robots in an unexpected way.

In both large-scale simulations, small numbers of robots could be observed wandering in small loops, circles or not moving at all. Upon the arrival of another agent, these non-productive cycles could be broken as the new agent wandered within the first's proximity field and the first executed avoidance.

Observation of these inactive agents showed that the non-productive cycles were the result of special physical conditions; robots trapped in closed circles of obstacles (including some memorable instances where other agents bricked them up), or special target object arrangements where repetitive "acquire", "drop", "acquire" loops could develop. These situations are analogous to the corners of the arena in the real-world experiments, where robots could become trapped if they approached the corner at a critical angle.

These problems, while not occurring with sufficient frequency to threaten the swarm as a whole, are probably due to the simplicity of the agent's responses. With their physical responses to stimuli limited to five discrete actions, some failure was probably inevitable and not too significant across the swarm as a whole. From these episodes we can conclude that the architecture would function best in a mildly dynamic environment. Low density swarms, or small swarms operating in environments where little independent movement can be expected will be more strongly affected, some continual noise in the environment is necessary to simulate the randomness created by fellow agents in a dense swarm.

Real-World

The real-world versions of this grouping task suffered from poor recognition. Three colours were about the limit of the vision system and the pillar object was frequently overlooked or accidentally seen in the reflections from the floor.

These false positives and negatives made finding the central pillar object problematic and encouraged the robots to disassemble nascent groups, exacerbating the problem observed in the small-scale simulations (see above), as in the real-world they struggled to re-assemble the groups. In general the real-world Grouping2 task emphasised the importance of wide, rapid searching (to find static objects) and good recognition.

4.5 General Observations

The following are some general observations made during the course of the experiments and implementation on factors effecting robot performance:

- Sensor field-of-view influences the robots' performance. Simulated robots with a very narrow field of view (below about 30 degrees) identified very few objects and spent much of their time searching. A very wide field-of-view (over about 90 degrees on these robots) caused the robots to identify

too many proximity objects and spend too much time avoiding. Clearly ideal sensor resolution is dependant on the sophistication of the actions and the hardware, there will be a complementary range between sophistication and execution for each design.

- Robots must be likely to encounter all relevant target objects in their local environment. The user cannot assume that robots travel far, even in a big environment. This idea was tested by placing all the pillars of the large-scale Grouping2 task in one corner, all the target objects in another and the robots in the middle. No target objects arrived at the pillars, event though the edges of the whole swarm overlapped both areas.

- Thresholds are less important than relevant objects. The relevance requirements and the limited number of objects likely to be known to any one individual at a time often means that correct behaviours are self-evident. When permitted to perform both large-scale Grouping1 and Grouping2 tasks in the same arena, robots overwhelmingly executed the Grouping1 task. This reflected the number of target objects (800), as opposed to the number of pillars (100) or robots (100). The threshold decreases for this behaviour, but it is also the easiest behaviour to implement.

Chapter 5

Future Work and Applications

"... departure will take place when flight-stores are complete. We are currently awaiting our full complement of small, lemon-soaked paper napkins for your comfort and convenience."

"Have you seen the world outside this ship? It's a wasteland, a desert! Civilisation's been and gone, there are no lemon-soaked paper napkins on their way from anywhere."

"The statistical probability is that other civilisations will arise. There will, one day, be small, lemon-soaked paper napkins. Until then, there will be a short delay."

— Douglas Adams,
The Hitchhiker's Guide to the Galaxy

Contents

The Results discussed in Chapter 4 demonstrate that The Architecture is operational in a controlled laboratory setting. We shall now discuss several improvements which would render The Architecture more robust in less structured "real-world" environments and lead to a publicly usable "release" version.

5.1 Developments

5.1.1 Thresholds and Stimulus

Section 3.4 discusses The Architecture's behaviour selection mechanism, inspired by the method of Theraulaz et al. (1998) (see Equation 2.1).

With the benefit of hindsight, the use of P as a ranking value is probably a misinterpretation of the scope of Theraulaz et al.'s (1998) behaviours. Realising behaviours in computer languages requires a very high degree of precision and consistency, a process which the author found led to a narrowing of scope. It is suspected that Theraulaz et al. (1998) would have intended for the stimulus and threshold selection process to operate in a vacuum, if a behaviour was not relevant. It would then have impelled the robot to search for the relevant components.

We took the design in a subtly different direction, employing as a behaviour selector the same basic form as Equation 2.1, but replacing s with relevance. To more elegantly implement their method (Equation 2.1), the following changes would be made to The Architecture:

- Re-introduce stimulus s by changing equation 3.4.1 to:

$$P_{i,j} = \frac{1}{2}(rel_{i,j} + \frac{s_{i,j}}{s_{i,j} + \theta_{i,j}}) \tag{5.1}$$

  As rel, s and θ are fixed between one and zero, equation 5.1 would always be between these values.

  Stimulus s would increment with time as in Theraulaz et al. (1998). $P_{i,j}$ would become the probability of robot i implementing behaviour j at a given time. A random number r would be compared to all values of P and a single behaviour randomly selected from all behaviours with a P value larger than r.

  This selection process would take place at regular intervals, implicitly enforcing a timeout.

- "Search" would become an implicit part of all behaviours, if the relevance of the selected behaviour was zero, the robot would engage the search behaviour. This would be calculated constantly. Thus the active behaviour would not be rigidly tied to relevance and robots may implicitly 'search for' objects.

It is believed that these modifications produce a more elegant system, with more emphasises on thresholds than the current design. However, it does not represent a major alteration.

5.1.2 Peer-to-Peer Communications

As discussed in Chapter 3 The Architecture does not include a direct analogue to chemical signals. This could be partially remedied by introducing peer-to-peer object sharing to allow knowledge of objects to propagate through the swarm.

This system would be a 'blind' process, with agents broadcasting their current object list to any other agents in proximity. Such transfer would allow the

swarm to act on the knowledge gained by widely separate agents. However, it would require several broad assumptions:

- The agents are in close physical proximity when the transfer occurs.
- Agents pass an object's data in close physical proximity to the observation point, or they can keep an accurate "dead-reckoning" account of their movement.
- Receiving agents have to translate directions from the transmitting agent's frame of reference to their own. Therefore, all agents have access to the same global directional reference – such as magnetic north, or a nest beacon.
- Only object bearing would be transmitted, agents would have to move "over there" to find objects. It would be expected that an agent's own sensors would perform finer localisation once in the object's vicinity.

Objects would need to be time-stamped to allow them to decay over time as they are transferred (thus preventing old and possibly false objects for cluttering an agent's object list). This would be implemented via a "local time since last observation" field, rather then a globally synchronised time-stamp.

5.1.3 Improved Perception

The current recognition of objects is clearly insufficient. Object detection on the basis of closely defined shapes and colours is a limitation.

To move beyond this, object recognition could be partially generalised by exploiting the size of the robot and its environment. Swarm robots are expected to be much smaller than their tasks, therefore any object resolvable from the robots' limited perspective is of a "nice" graspable size. On this scale primitive shapes (round, not-round, rectangle, etc.) and crude colour distinctions (plant-green, not-plant-green, etc.) could be employed. This approach is far less specific than the current system.

The ambiguity of real-world sensors could be partially countered by "fuzzifying" object type membership, making an object a member of each type to a degree and reflecting this in relevance calculation. Degrees of membership could also improve identification of composite objects such as groups. A group with more objects would rate higher as a member of type "group".

This would improve decision-making in the presence of similar objects.

The localisation of objects could be rendered more general by removing the requirement for exact bearings and ranges and replacing them with a "close", "far", "left", "right" categorisation. This would help to divorce the object localisation process from hardware specifics such as camera focal-length. Such vague positioning could be based upon perceived object size, assuming that all objects in the agent's small field are of a similar "nice" size.

Experience with real-world hardware would also suggest that robots would benefit from a more accurate picture of its own movement. Sensors such as odometers, and gripper feedback would enable better action implementation. Accessing these sensors would be an matter internal to the actions themselves and does not impinge on The Architecture as a whole.

5.1.4 New Agents

Castes

Provided that agent physical design obeys the model of a swarm agent discussed in Section 2.1 there are no firm restrictions on the physical appearance of an agent.

A agent "caste" would be a design focused on a smaller sub-set of swarm tasks. Examples of possible castes are:

- "Scout" – a design emphasising locomotion and perception over manipulation. Such an agent might be fast (such as a quad-copter), with several powerful cameras but no pincers. Instead, it could permanently search for objects and spread information through the swarm by means of peer-to-peer communication.
- "Storage" – an agent with battery or solar cell. Such a caste could be slow or even immobile with grippers replaced with charge contacts.
- "Excavator" – a caste with a powerful or precise manipulator for heavy or fine work. This could be at the expense of perception, with this caste relying on other agents to scout its world.

The majority of the swarm would still be the basic "worker" caste.

Notice that these four ideas do not deviate from locomotion, perception and manipulation fundamentals (see Chapter 3). New castes should not introduce specialised hardware that cannot be handled by the original four behaviour actions – the bare minimum of moving parts is still assumed. For instance, a drilling caste would mount a drill instead of a manipulator and use the "grip" and "un-grip" actions to engage the drill. Specialised castes are a *reduction* of generality, not an expansion of the base concepts.

Physical Design

The real-world robots used in this project were not overly sophisticated, even for swarm robots. Regardless of caste, the next generation of robots could incorporate several design improvements without becoming more than swarm agents:

- *Internal Sensors* – The need for a universal reference bearing has already been mentioned in relation to communications. A magnetic compass could serve this ambition.

  Feedback from other internal sensors, such as odometers and an accelerate (to measure robot 'tilt') could be incorporated into the action control loops and provide better response.

- *Improved Manipulation* – The real-world robots' "scoop" gripper is clearly inadequate, a more sensitive gripper with better tactile sensing (more reliable assessment of "captured-objects") and better articulation would be a great improvement. This could include more articulated "finger" joints, to better grip objects and more degrees of freedom. This last would be essential if the robots are to "stack" objects in three-dimensions, as would be needed for a complex building task.

Advanced manipulators would still focus on one object at a time and obey the "drop" and "grasp" actions exclusively.

5.1.5 Other Survival Actions

This project only implements one survival action: avoid. A more polished version of The Architecture should also include a "recharge" action. This would imply the need for a "re-charge-station" object and battery monitoring hardware.

The recharge action would simply be activated when the battery charge falls below a hard-coded threshold and would consist of "moving-to" the re-charge station object. Docking with the station would be integral to this action.

As avoidance would still need to be usable during the re-charge action, execution of survival actions would need to be sensitive to avoidance and set the re-charge-station object as an exception.

This action was not implemented in this version of The Architecture as it was felt to be a time and resource consuming distraction to the main project.

5.1.6 User Control

The behaviours in both the simulated and real-world robots were in actual fact hard-coded into The Architecture's executable program. This was to save time, as a universal, user-friendly behavioural system would function exactly the same and neither version of The Architecture was intended for general release. Release versions of The Architecture would need to take a more universal approach.

The author does not believe that the behaviour creation process should be formalised as a language. The process is not designed to be extensible – for example; new user defined actions are not expected.

Instead, the best option is probably a graphical interface that defines which actions and relevant objects are needed for each behaviour. The configuration data generated from such an interface would be very small.

New behaviours could be disseminated to agents by several methods:

- General Broadcast – requiring all agents to be sensitive to some global channel. The easiest method to implement, but the one most opposed to the swarm ethos.
- Upon Recharge – all agents must eventually recharge their batteries. The re-charge stations could also serve as dissemination points for new behaviours. This is a more decentralised approach.
- By Agent Interaction – New behaviours could be disseminated by a peer-to-peer mechanism, similar to that discussed above for objects. This is the most "swarm-like" and could be combined with the either of the two above approaches.

5.2 Applications

The Architecture is intended for swarm-like tasks only. As discussed in Section 1.3 these are tasks that do not require tight precision from the agents.

Good examples would be:

- *Cleaning* – ordering randomly dropped debris, either in piles or at a depot are variations on the grouping tasks.

  Examples of this type of task are:

  - Floor cleaning.
  - Contaminant removal – such as weeds from a water supply.
  - Removal of dirt from sensitive plant.

  Cleaning is a specific case of foraging.

- *Foraging* – a classic swarm task. Collecting objects of interest in a dynamic environment would be essential for a large swarm and could conceivably include:

  - Construction materials, such as bricks, stones, etc.
  - Collection of the raw materials for manufacturing useful objects, such as building materials, structures or other agents. Rapid-prototyping technologies, such as 3D-printing, could be employed to manufacture useful objects.

- *Construction* – building along the lines discussed in Chapter 2 (or in Dorigo et al. (2000), Turner (2011), Franks & Deneubourg (1997)), small, repeating units being brought together to form a larger assembly. Examples of such constructions could be:

  - Earth or natural bunkers around sensitive plant.
  - Paths or carriageways through recurring growth.
  - Unpacking and moving/assembling modular plant components.

A large swarm could attempt all of these tasks simultaneously, by assigning each task to a smaller sub-swarm. The ultimate objective would be to create a self-organising colony that produces, assembles or maintains useful artefacts.

Ultimately, it must be remembered that swarm agents are insect-like. They cannot be taxed with over-complex tasks and are not likely to be the "best" (most efficient, most precise, etc.) solution. They are likely to be most useful in environments where more complex agents cannot be deployed, such as:

- Space, the moon, asteroids, etc.
- Underwater.
- Inhospitable urban zones, such as pipes, conduits, etc.
- Underground.
- Inhospitable terrain, such as deserts, Antarctica, mountains, jungle, etc.
- Hazardous environments, such as environments with extremes of heat, cold, radiation, presence of caustics, high or low pressure, etc.

Chapter 6

Conclusions

> . . . huh? oh sure. . . why does the human want dried leaves in boiling water? Answer: because he's an ignorant monkey that doesn't know any better! Cute, huh?
>
> — Douglas Adams,
> *The Hitchhiker's Guide to the Galaxy*

Contents

The Architecture functions on simple, idealised swarm tasks upon both real-world and simulated swarms. The simulated swarm shows a high-degree of scalability. Architecture controlled swarms can cooperate on several swarm-tasks without fundamental modification of their control programs.

The agents controlled by The Architecture are not complete insects, they have fewer behaviours and their focus on visual identification is not quite compatible with insects chemical markers (see Section 3.2.3). However, key aspects of swarm cooperation have been identified and these components could be added without major revision of The Architecture (see Sections 3.2.3 and 5.1.3).

6.1 Implementation

The following points are important in Architecture implementation:

- Unambiguous object recognition is critical. The real-world sensors were noisy and inexact in localisation, this was far less important than accurate recognition of objects.

- Agents need to distinguish between free objects and objects captured by other agents if they are not to work against each other.
- Agents need to distinguish between atomic objects and objects that form part of a larger complete or semi-complete structure. See Section 4.3.4.
- It can be expected that the above points will fail for some interactions and some agents. A certain amount of agent failure is inevitable.
- The Architecture is capable of facilitating emergent behaviour – collectively, swarms may achieve objectives that are not explicitly coded, such as the division of labour in the ball-passing task and the location of groups in Grouping1.
- Robots will "specialise" in whatever behaviour is most probable. Be careful that behaviour criteria are not set so one behaviour pre-dominates.
- A minimum "agent density" exists for each task and environment. The agents must be encouraged to interact by environmental constraints and be sufficiently likely to encounter one another.
- Like sensing, gripping must be reasonably effective. The agents should be able to keep control of objects despite environmental considerations. They should be mechanically simple, but also need to be effective.

6.2 The Architecture and The Swarm Ethos

The following is a brief discussion of The Architecture in the context of the Swarm Ethos, discussed in Chapter 1. We shall answer each point in the ethos, comparing the final Architecture design to the original description of a swarm.

1. Physically small. The real-world robots are reasonably small and inexpensive. With the continuing development of embedded computers, it is expected that much smaller machines with equivalent computational power could be built.
2. Mechanical simplicity. The Architecture only supports five basic behavioural actions and two sets of moving parts: the gripper and the locomotion motors – both of these are expected to be minimalist.
3. Local interactions and local peer-to-peer communication only. At the current time, The Architecture is deaf and mute, although as Section 5.1.2 illustrates, it would also be amenable to local peer-to-peer, low-bandwidth data-transfer.
4. Perception is local only and may be subject to error and misinterpretations. As has been discussed, The Architecture needed to be made robust to the noise in real-world sensors – see Section 3.2.2 for detail on object confidence – but perception is utterly local and, in practice, noisy.
5. No maps, plans etc. As discussed in the previous chapter, such abstractions are not part of The Architecture, which can drive a swarm to tasks beyond the comprehension of its individual members.

6. Global cooperation is an illusion that exists only in the eye of the (human) observer. See above point.

7. Swarm agents are cheap and disposable as individuals. The real-world robots were neither overly expensive or mechanically sophisticated.

8. Swarm agents are fallible. Examples in Sections 4.5, 4.4.4, 4.3.4, 4.2.4 provide ample evidence of this.

Thus we conclude that The Architecture conforms to our definition of a swarm.

6.3 The Architecture Scope

The Architecture also compares well to the original specifications discussed in Chapter 1. The Architecture has fulfilled the original requirements and, as shown previously in chapter 5, laid the foundations for far more sophisticated developments.

6.3.1 Biological Inspirations

While more limited than real-world insects (see recruitment, above) The Architecture displays a number of the key features of insect swarms:

- Stigmergic cooperation. As discussed in Chapter 3, all decisions are ultimately based on observed objects, thus cooperation among the swarm is the "emergent result of the collective dynamics of either interacting autonomous agents or basic control units in a single agent..." (Trianni et al. 2011) as discussed in Section 2.3.

- Division of Labour. As demonstrated by the ball-passing task, The Architecture is able to divide the work among the swarm on a roughly equal basis. This is implicit in the behaviour selection and is not explicitly coded for.

  See Section 4.2 for results details.

- Emergent behaviour. As discussed in Section 4.2.4 a primitive recruitment emerges from the "receive" behaviour.

- Tasks beyond the individual. As discussed in Section 4.3.4 the large-scale simulations demonstrate that the individual robots can cooperate on tasks as a group that they cannot conceive of as individuals.

6.3.2 Task Agnosticism

The Architecture demonstrated its task agnosticism by functioning on three different (although related) swarm-tasks. Although these tasks were not radically different in concept, only minor changes needed to be made to the robots to effect quite different overall swarm results. The simplicity of task modification has the potential for multi-task swarms, as described in Section 5.1.1.

6.3.3 Hardware Agnosticism

The differences between the simulation and the real-world robots; their different physical appearance and their differing interactions with their environment, illustrate that The Architecture can operate on more than one swarm robot design. This fundamental requirement of generalisation is demonstrated by the comparable results in Chapter 4 and gives the possibility of multi-caste swarms, as discussed in Section 5.1.4.

6.3.4 Swarm Size Indifference

The flexibility of The Architecture with respect to swarm size is demonstrated by the successful results of the large and small-scale grouping tasks (see Section 4.3 and 4.4). The swarms may be arbitrarily scaled from four to one hundred robots without controller modification. The only caveat is that there must be enough work for all members of the swarm.

Bibliography

Adouane, L. & Le Fort-Piat, N. (2004), Hybrid behavioral control architecture for the cooperation of minimalist mobile robots, *in* 'Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on', Vol. 4, pp. 3735 – 3740 Vol.4.

Agre, P. E. & Chapman, D. (1987), Pengi: An implementation of a theory of activity, *in* 'Proceedings of the Sixth National Conference on Artificial Intelligence - Volume 1', AAAI'87, AAAI Press, pp. 268–272.

Alfredo, W., Arkin, R. C., Cervantes-Perez, F., Olivares, R. & Corbacho, F. (1998), A neural schema architecture for autonomous robots, *in* 'Int. Symposium on Robotics and Automation', pp. 12–14.

Arkin, R. C. (1992), 'Cooperation without communication: Multiagent schema-based robot navigation', *Journal of Robotic Systems* **9**(3), 351–364.

Arkin, R. C. (1998), *Behavior-Based Robotics*, MIT Press.

Arkin, R. C. & Balch, T. (1997), 'Aura: principles and practice in review', *Journal of Experimental & Theoretical Artificial Intelligence* **9**(2-3), 175–189.

Baldassarre, G., Trianni, V., Bonani, M., Mondada, F., Dorigo, M. & Nolfi, S. (2007), 'Self-organized coordinated motion in groups of physically connected robots', *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on* **37**(1), 224 –239.

Bekey, G. A. (2005), *Autonomous Robots, From Biological Inspiration to Implementation and Control*, MIT Press.

Bodi, M., Thenius, R., Szopek, M., Schmickl, T. & Crailsheim, K. (2012), 'Interaction of robot swarms using the honeybee-inspired control algorithm beeclust', *Mathematical and Computer Modelling of Dynamical Systems* **18**(1), 87–100.

Bojadziev, G. & Bojadziev, M. (1995), *Fuzzy Sets, Fuzzy Logic, Applications*, Vol. 5 of *Advances in Fuzzy Systems — Applications and Theory*, World Scientific.

Bradski, G. & Kaehler, A. (2008), *Learning OpenCV – Computer Vision With The OpenCV Library*, O'Reilly Media.

Brambilla, M., Ferrante, E., Birattari, M. & Dorigo, M. (2013), 'Swarm robotics: a review from the swarm engineering perspective', *Swarm Intelligence* **7**(1), 1–41.

Brooks, R. A. (1999), *Cambrian Intelligence, The Early History of the New AI*, The MIT Press.

Brooks, R. A. (2002), *Robot: The Future of Flesh and Machines*, Penguin Books.

Buck, S., Schmitt, T. & Beetz, M. (2002), Reliable multi-robot coordination using minimal communication and neural prediction, *in* M. Beetz, J. Hertzberg, M. Ghallab & M. Pollack, eds, 'Advances in Plan-Based Control of Robotic Agents', Vol. 2466 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 157–190. 10.1007/3-540-37724-7_3.

Caprari, G. & Siegwart, R. (2005), Mobile micro-robots ready to use: Alice, *in* 'Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on', pp. 3295 – 3300.

Castelfranchi, C. (2006), Silent agents: From observation to tacit communication, *in* J. Sichman, H. Coelho & S. Rezende, eds, 'Advances in Artificial Intelligence - IBERAMIA-SBIA 2006', Vol. 4140 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 98–107. 10.1007/11874850_14.

Chapman, R. F. (1998), *The Insects: Structure and Function*, 4 edn, Cambridge University Press.

Cicirello, V. & Smith, S. (2001), Ant colony control for autonomous decentralized shop floor routing, *in* 'Autonomous Decentralized Systems, 2001. Proceedings. 5th International Symposium on', pp. 383 –390.

Şahin, E. (2005), Swarm robotics: From sources of inspiration to domains of application, *in* E. Şahin & W. Spears, eds, 'Swarm Robotics', Vol. 3342 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 10–20. 10.1007/978-3-540-30552-1_2.

Deag, J. M. (1980), *Social behaviour of Animals*, number 118 *in* 'Studies in Biology', Edward Arnold Ltd.

Deneubourg, J. & Goss, S. (1989), 'Collective patterns and decision-making', *Ethology Ecology & Evolution* **1**(4), 295–311.

Detrain, C. & Pasteels, J. M. (1992), 'Caste polyethism and collective defense in the ant, *pbeidole pallidula*;: the outcome of quantitative differences in recruitment', *Behavioral Ecology and Sociobiology* **29**, 405–412. 10.1007/BF00170170.

Dorigo, M., Bonabeau, E. & Theraulaz, G. (2000), 'Ant algorithms and stigmergy', *Future Generation Computer Systems* **16**(8), 851 – 871.

Drickamer, L. C., Vessey, S. H. & Jakob, E. M. (1996), *Animal Behavior: Mechanisms, Ecology, Evolution*, 5th edition edn, McGraw-Hill Higher Education.

Duarte, M., Christensen, A. & Oliveira, S. (2011), Towards artificial evolution of complex behaviors observed in insect colonies, *in* 'Progress in Artificial Intelligence', Vol. 7026 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 153–167. 10.1007/978-3-642-24769-9_12.

Dugatkin, L. A. & Reeve, H. K., eds (1998), *Game Theory and Animal Behavior*, Oxford University Press.

Florczyk, S. (2005), *Robot Vision, Video-based Indoor Exploration with Autonomous and Mobile Robots*, Wiley-VCH.

Franks, N. R. & Deneubourg, J.-L. (1997), 'Self-organizing nest construction in ants: individual worker behaviour and the nest's dynamics', *Animal Behaviour* **54**(4), 779 – 796.

Fujisawa, R., Dobata, S., Sugawara, K. & Matsuno, F. (2014), 'Designing pheromone communication in swarm robotics: Group foraging behavior mediated by chemical substance', *Swarm Intelligence* **8**(3), 227–246.

Garnier, S., Gautrais, J. & Theraulaz, G. (2007), 'The biological principles of swarm intelligence', *Swarm Intelligence* **1**, 3–31. 10.1007/s11721-007-0004-y.

Garnier, S., Jost, C., Gautrais, J., Asadpour, M., Caprari, G., Jeanson, R., Grimal, A. & Theraulaz, G. (2008), 'The embodiment of cockroach aggregation behavior in a group of micro-robots', *Artificial Life* **14**(4), 387–408.

Garnier, S., Jost, C., Jeanson, R., Gautrais, J., Asadpour, M., Caprari, G. & Theraulaz, G. (2005), Collective decision-making by a group of cockroach-like robots, *in* 'Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE', pp. 233 – 240.

Goldberg, D. (2001), Evaluating the Dynamics of Agent-environment Interaction, PhD thesis, School of Engineering, Los Angeles, CA, USA.

Gordon, D. M. (1999), *Ants at Work, How an Insect Society Is Organised*, Simon & Schuster Inc.

Grassé, P.-P. (1959), 'La reconstruction du nid et les coordinations interindividuelles chezbellicositermes natalensis etcubitermes sp. la théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs', *Insectes sociaux* **6**(1), 41–80.

Gullan, P. J. & Cranston, P. S. (1994), *The Insects; An Outline of Entomology*, Chapman & Hall.

Hecht, E. & Zjac, A. (1974), *Optics*, Addison-Wesley.

Holland, O. & Melhuish, C. (1999), 'Stigmergy, self-organization, and sorting in collective robotics', *Artif. Life* **5**(2), 173–202.

Hölldobler, B. K. & Wilson, E. O. (1989), 'Paths of glory: Following army ants as they march through the tropics.', *Sciences* **29**(6), 18.

Hölldobler, B. & Wilson, E. O. (1995), *Journey to the Ants: a story of scientific exploration*, Harvard University Press.

Howse, P. E. (1970), *Termites: A Study in Social Behaviour*, Biological Sciences, Hutchinson and Co.

Hu, M.-K. (1962), 'Visual pattern recognition by moment invariants', *Information Theory, IRE Transactions on* **8**(2), 179–187.

Huntsberger, T., Aghazarian, H., Baumgartner, E. & Schenker, P. (2000), Behavior-based control systems for planetary autonomous robot outposts, *in* 'Aerospace Conference Proceedings, 2000 IEEE', Vol. 7, pp. 679 –686 vol.7.

Huntsberger, T., Pirjanian, P., Trebi-Ollennu, A., Das Nayar, H., Aghazarian, H., Ganino, A., Garrett, M., Joshi, S. & Schenker, P. (2003), 'Campout: a control architecture for tightly coupled coordination of multirobot systems for planetary surface exploration', *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on* **33**(5), 550 – 559.

Ijspeert, A., Martinoli, A., Billard, A. & Gambardella, L. (2001), 'Collaboration through the exploitation of local interactions in autonomous collective robotics: The stick pulling experiment', *Autonomous Robots* **11**, 149–171. 10.1023/A:1011227210047.

Ishiwata, H., Noman, N. & Iba, H. (2011), Emergence of cooperation in a bio-inspired multi-agent system, *in* J. Li, ed., 'AI 2010: Advances in Artificial Intelligence', Vol. 6464 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 364–374. 10.1007/978-3-642-17432-2_37.

Isik, M., Stulp, F., Mayer, G. & Utz, H. (2007), Coordination without negotiation in teams of heterogeneous robots, *in* G. Lakemeyer, E. Sklar, D. Sorrenti & T. Takahashi, eds, 'RoboCup 2006: Robot Soccer World Cup X', Vol. 4434 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 355–362. 10.1007/978-3-540-74024-7_33.

Kallel, I., Chatty, A. & Alimi, A. (2008), Self-organizing multirobot exploration through counter-ant algorithm, *in* K. Hummel & J. Sterbenz, eds, 'Self-Organizing Systems', Vol. 5343 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 133–144. 10.1007/978-3-540-92157-8_12.

Kernbach, S., ed. (2013), *Handbook of Collective Robotics: Fundamentals and Challanges*, CRC Press Taylor & Francis Group.

Kok, J. R., Spaan, M. T. & Vlassis, N. (2005), 'Non-communicative multi-robot coordination in dynamic environments', *Robotics and Autonomous Systems* **50**(23), 99 – 114. Multi-Robots in Dynamic Environments.

Labella, T. H., Dorigo, M. & Deneubourg, J.-L. (2006), 'Division of labor in a group of robots inspired by ants' foraging behavior', *ACM Trans. Auton. Adapt. Syst.* **1**(1), 4–25.

Lee, G., Chong, N. Y. & Defago, X. (2007), Robust self-deployment for a swarm of autonomous mobile robots with limited visibility range, *in* 'Robot and Human interactive Communication, 2007. RO-MAN 2007. The 16th IEEE International Symposium on', pp. 925 –930.

Lerman, K. & Galstyan, A. (2002), 'Mathematical model of foraging in a group of robots: Effect of interference', *Autonomous Robots* **13**(2), 127–141.

Liu, W., Winfield, A. F. T., Sa, J., Chen, J. & Dou, L. (2007), 'Towards energy optimization: Emergent task allocation in a swarm of foraging robots', *Adaptive Behavior* **15**(3), 289–305.

Lötzsch, M., Bach, J., Burkhard, H.-D. & Jüngel, M. (2004), Designing agent behavior with the extensible agent behavior specification language xabsl, *in* D. Polani, B. Browning, A. Bonarini & K. Yoshida, eds, 'RoboCup 2003: Robot Soccer World Cup VII', Vol. 3020 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 114–124.

Martinoli, A., Easton, K. & Agassounon, W. (2004), 'Modeling swarm robotic systems: a case study in collaborative distributed manipulation', *The International Journal of Robotics Research* **23**(4-5), 415–436.

Matarić, M. J. (1994), Interaction and Intelligent Behavior, PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.

Matthey, L., Berman, S. & Kumar, V. (2009), Stochastic strategies for a swarm robotic assembly system, *in* 'Robotics and Automation, 2009. ICRA '09. IEEE International Conference on', pp. 1953–1958.

McGavin, G. C. (2001), *Essential Entomology, an Order-by-Order Introduction*, Oxford University Press.

Merkle, D. & Middendorf, M. (2004), 'Dynamic polyethism and competition for tasks in threshold reinforcement models of social insects', *Adaptive Behavior* **12**(3-4), 251–262.

Min, H. & Wang, Z. (2010), Group escape behavior of multiple mobile robot system by mimicking fish schools, *in* 'Robotics and Biomimetics (ROBIO), 2010 IEEE International Conference on', pp. 320 –326.

Mondada, F., Gambardella, L., Floreano, D., Nolfi, S., Deneuborg, J.-L. & Dorigo, M. (2005), 'The cooperation of swarm-bots: physical interactions in collective robotics', *Robotics Automation Magazine, IEEE* **12**(2), 21–28.

Nouyan, S., Gross, R., Bonani, M., Mondada, F. & Dorigo, M. (2009), 'Teamwork in self-organized robot colonies', *Evolutionary Computation, IEEE Transactions on* **13**(4), 695 –711.

Pagello, E., DAngelo, A., Montesello, F., Garelli, F. & Ferrari, C. (1999), 'Cooperative behaviors in multi-robot systems through implicit communication', *Robotics and Autonomous Systems* **29**(1), 65 – 77.

Parker, C. A. C. & Zhang, H. (2006), 'Collective robotic site preparation', *Adaptive Behavior* **14**(1), 5–19.

Parker, L. (1998), 'Alliance: an architecture for fault tolerant multirobot cooperation', *Robotics and Automation, IEEE Transactions on* **14**(2), 220 –240.

Parker, L. E. (2008), Multiple mobile robot systems, *in* B. Siciliano & O. Khatib, eds, 'Springer Handbook of Robotics', Springer Berlin Heidelberg, pp. 921–941. 10.1007/978-3-540-30301-5_41.

Payton, D., Estkowski, R. & Howard, M. (2005), Pheromone robotics and the logic of virtual pheromones, *in* E. ahin & W. Spears, eds, 'Swarm Robotics', Vol. 3342 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 45–57.

Phan, T. A. & Russell, R. A. (2012), 'A swarm robot methodology for collaborative manipulation of non-identical objects', *The International Journal of Robotics Research* **31**(1), 101–122.

Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G., Ducatelle, F., Birattari, M., Gambardella, L. M. & Dorigo, M. (2012), 'ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems', *Swarm Intelligence* **6**(4), 271–295.

Pirjanian, P. (2000), 'Multiple objective behavior-based control', *Robotics and Autonomous Systems* **31**(12), 53 – 60.

Pirjanian, P., Huntsberger, T. L., Trebi-Ollennu, A., Aghazarian, H., Das, H., Joshi, S. S. & Schenker, P. S. (2000), CAMPOUT: a control architecture for multirobot planetary outposts, *in* G. T. McKee & P. S. Schenker, eds, 'Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series', Vol. 4196 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pp. 221–230.

Rubenstein, M., Ahler, C. & Nagpal, R. (2012), Kilobot: A low cost scalable robot system for collective behaviors, *in* 'Robotics and Automation (ICRA), 2012 IEEE International Conference on', pp. 3293–3298.

Rubenstein, M., Cabrera, A., Werfel, J., Habibi, G., McLurkin, J. & Nagpal, R. (2013), Collective transport of complex objects by simple robots: Theory and experiments, *in* 'Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems', AAMAS '13, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, pp. 47–54.

Rubenstein, M., Cornejo, A. & Nagpal, R. (2014), 'Programmable self-assembly in a thousand-robot swarm', *Science* **345**(6198), 795–799.

Ruiz, M. & Uresti, J. (2008), Team agent behavior architecture in robot soccer, *in* 'Robotic Symposium, 2008. LARS '08. IEEE Latin American', pp. 20 –25.

Schmickl, T. & Crailsheim, K. (2008*a*), An individual-based model of task selection in honeybees, *in* M. Asada, J. Hallam, J.-A. Meyer & J. Tani, eds, 'From Animals to Animats 10', Vol. 5040 of *Lecture Notes in Computer*

Science, Springer Berlin / Heidelberg, pp. 383–392. 10.1007/978-3-540-69134-1_38.

Schmickl, T. & Crailsheim, K. (2008*b*), 'Taskselsim: a model of the self-organization of the division of labour in honeybees', *Mathematical and Computer Modelling of Dynamical Systems* **14**(2), 101–125.

Schmickl, T., Möslinger, C. & Crailsheim, K. (2007), Collective perception in a robot swarm, *in* E. ahin, W. Spears & A. F. Winfield, eds, 'Swarm Robotics', Vol. 4433 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 144–157.

Schmitt, T., Hanek, R., Beetz, M., Buck, S. & Radig, B. (2002), 'Cooperative probabilistic state estimation for vision-based autonomous mobile robots', *Robotics and Automation, IEEE Transactions on* **18**(5), 670 – 684.

Stroupe, A., Okon, A., Robinson, M., Huntsberger, T., Aghazarian, H. & Baumgartner, E. (2006), 'Sustainable cooperative robotic technologies for human and robotic outpost infrastructure construction and maintenance', *Autonomous Robots* **20**, 113–123. 10.1007/s10514-006-5943-4.

Stulp, F., Isik, M. & Beetz, M. (2006), Implicit coordination in robotic teams using learned prediction models, *in* 'Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on', pp. 1330 –1335.

Sudd, J. H. (1970), *An Introduction to the Behaviour of Ants*, Edward Arnold, London.

Terano, T., Asai, K. & Sugeno, M. (1992), *Fuzzy Systems Theory and Its Applications*, Academic Press.

Thalmann, D. & Musse, S. R. (2007), *Crowd Simulation*, Springer.

Theraulaz, G., Bonabeau, E. & Denuebourg, J.-N. (1998), 'Response threshold reinforcements and division of labour in insect societies', *Proceedings of the Royal Society of London. Series B: Biological Sciences* **265**(1393), 327–332.

Trianni, V., Tuci, E., Passino, K. & Marshall, J. (2011), 'Swarm cognition: an interdisciplinary approach tothestudy of self-organising biological collectives', *Swarm Intelligence* **5**, 3–18. 10.1007/s11721-010-0050-8.

Turner, J. (2011), 'Termites as models of swarm cognition', *Swarm Intelligence* **5**, 19–43. 10.1007/s11721-010-0049-1.

Wagner, I. A., Altshuler, Y., Yanovski, V. & Bruckstein, A. M. (2008), 'Cooperative cleaners: A study in ant robotics', *The International Journal of Robotics Research* **27**(1), 127–151.

Werfel, J., Bar-Yam, Y., Rus, D. & Nagpal, R. (2006), Distributed construction by mobile robots with enhanced building blocks, *in* 'Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on', pp. 2787 –2794.

Werger, B. B. & Matarić, M. J. (2000), Broadcast of local eligibility for multi-target observation, *in* L. Parker, G. Bekey & J. Barhen, eds, 'Distributed Autonomous Robotic Systems 4', Springer Japan, pp. 347–356.

Wilson, E. (1976), 'The organization of colony defense in the ant pheidole dentata mayr (hymenoptera: Formicidae)', *Behavioral Ecology and Sociobiology* **1**(1), 63–81.

Wilson, E. O. (1979), *The Insect Societies*, Harvard University Press.

Wilson, E. O. & Hölldobler, B. (2005), 'Eusociality: origin and consequences', *Proceedings of the National Academy of Sciences of the United States of America* **102**(38), 13367–13371.

Zhang, D., Xie, G., Yu, J. & Wang, L. (2007), 'Adaptive task assignment for multiple mobile robots via swarm intelligence approach', *Robotics and Autonomous Systems* **55**(7), 572 – 588.

Appendix A

Relevant Publications

Proceedings of Australasian Conference on Robotics and Automation, 2-4 Dec 2013, University of New South Wales, Sydney Australia

A Cooperative Architecture Based on Social Insects

Iain Brookshaw, Dr. Tobias Low
University of Southern Queensland
w0086292@umail.usq.edu.au

October 31, 2013

Abstract

In the last two decades, cooperative robotic groups have advanced rapidly; beginning with simple, almost blind box pushing tasks and advancing to the complexity of robocup's autonomous soccer matches. Groups of machines have been employed to build structures, search for targets, mimic insects and enact complex formations with precision and aplomb. The complexity of the tasks accomplished have been both impressive and practical, clearly illustrating the potential power of robotic groups and demonstrating how they may be applied to solve real-world problems.

Building on this success, we have created a software architecture that was intended to remove the robotic agents' dependency on complex communications or detailed task specific information. By incorporating biological models of stigmergic social insect cooperation into the architecture, we aim to ensure that the robots will be able to cooperate implicitly, without regard to group size and with only a weak dependency on task specific information and group homogeneity.

We have conducted preliminary investigations into the design's feasibility by using computer simulations of a simple object passing task. This simple task has enabled us to establish that cooperation is possible using this system. This paper will discuss the system's origins, design and future expansion.

1 Introduction

Given that cooperation among simple agents is more desirable than a single complex machine, it follows that a cooperative system should be as general as possible, without being wedded to a single objective or based on strong assumptions about other robots.

Many cooperative projects are based around one goal: to play soccer [Pagello *et al.*, 1999], to sort or group objects [Holland and Melhuish, 1999], mimic an aspect of insect behaviour [Phan and Russell, 2012], to pull sticks or pegs [Ijspeert *et al.*, 2001] and so on. Each of these tasks may possess a number of components or sub-tasks, implemented through a number of behaviours, but usually the groups' principle objective is explicitly hand coded. This means that any minor change in objective usually implies starting from scratch. Thus in may projects, it would be difficult to add new behaviours without rebuilding the whole decision making process.

There are several systems that have addressed this, creating broader software structures that permit multiple tasks to be attempted. However, many of these employed direct, explicit communications between agents. We will address the disadvantages of this in more detail in Section 2, but the basic objections are the limitations on group size, the strong requirement of homogeneity and the fragility of radio-based networks.

We believe that these limitations are too restrictive and present an architecture aimed at overcoming them. We began with the biological inspiration of cooperative insects, employing a model of stigmergic hive cooperation that negates the need for both central control and explicit cooperation (although insects do use explicit communication channels, we focused on their implicit methods. See Section 3 for details). By combining this concept with ideas taken from behaviour-based architectures of previous experiments we believe that we have created a system that has the potential to be fully general and not limited by group size, homogeneity or task.

In general, we sought to include the following key points in our design:

- Agents should not explicitly communicate. This is a restriction in both hardware homogeneity (necessitating transmit and receive units and protocols) and group size as well as being unnecessary.
- Agents should not need to be aware of a 'global' world model. Each agent should be able to cooperate with the others based only on what it alone can 'see' at any given time. Long memories, maps and global world models are not needful.
- Exact control is unnecessary. Because this is a group architecture, it is not necessary to direct a robot to do *that* job *there* and *then*. Instead, it can be assumed that all tasks will be addressed by a member of the group at some point. By taking advantage of the distributed nature of cooperative groups, the definite article has been discarded in favour of more generality.
- Robots should be easily re-programmable for a variety of tasks without modification to their decision making mechanism. Behaviours should form basic interchangeable structures that provide the tools necessary for the robots to solve a task.

  Ultimately, tasks should be assigned by another computer. While this is beyond the scope of this project, the design should reflect this by clearly defining the rules for behaviours.

These ideas have been implemented in a simple simulated task.

2 Past Cooperative Robots

When building cooperative groups it is imperative that they possess some form of communication. In some fashion, all cooperative agents have some means of coordinating their actions with others in the group.

In general, there are two ways of doing this: explicitly, by broadcasting their desired and intentions and implicitly, by observing the actions of others. As defined in Kernbach [2013] pg 120: "In explicit cooperation, robots elaborate locally the preferences for their behaviours, based on local, eventually shared, world models." Castelfranchi [2006] describes implicit cooperation as a communication method where "[the] practical behaviour itself is the message," where observation is the foundation of cooperation.

Many cooperative projects employ explicit communications to simplify the cooperative task. Schmitt *et al.* [2002] and Montijano *et al.* [2011] use explicitly linked groups to improve localisation, while Bekey *et al.* [2011], Arkin [1998] and Schwertfeger and Jenkins [2007] describe means by which groups may employ communications to engage in collective decision making.

By contrast other projects employ communications to enable individual robots to make decisions on their own. This decentralised approach removes a central controller by giving all robots a finite list of behaviours. Robots select the correct behaviour based on the information it can perceive, input from other individuals in the swarm and an internal decision making mechanism. This approach has its origins in the work of Brooks [Brooks, 2002]. In cooperative robotics the ALLIANCE system of Parker [1998] is probably the most widely reported. Other general, decentralised, behaviour-based architectures include the early 'nerd herd' of Matarić [1997] and the CAMPOUT architecture of Huntsberger *et al.* [2003]. In all of these explicit radio communications are a factor in the design.

However, explicit communications have been criticised as being power intensive, fragile and (most critically), difficult to scale arbitrarily [Şahin, 2005]. Agent groups cannot usually be easily supplemented in an ad hoc fashion and so this approach is limited in a real world environment [Agmon and Stone, 2011].

Despite the common usage of explicit communications, some elegant systems have been created that employ fully or partial implicit observation. The work of Ijspeert *et al.* [2001] is often cited as one of the most significant. In this elegant experiment, an number of very limited robots cooperated to pull a series of sticks from matching holes in the table-top. Each robot was too limited to remove the entirety of the stick by itself, but relied on other robots observing its difficulty and coming to its aid.

The stick pulling project seems simplistic, but it helped establish an important principle; that robots can successfully attempt manipulative tasks without the need for dense communication. This has been reinforced by a number of experiments: Kok *et al.* [2005] created a robot-soccer team based on implicit cooperation and 'locker-room' agreements, Pagello *et al.* [1999] also produced a behaviour-based robotic soccer team using implicit cooperation. Behaviours were selected by and arbitration module "... hand-coded by intensive use of heuristic from soccer domain experience...". Phan

and Russell [2012] successfully attempted to reproduce weaver ant nest building behaviour by employing a group of limited robots to curl a rubber "leaf". Holland and Melhuish [1999] used stigmergic ant sorting behaviour (ants are able to sort different classes of young into groups) to create a hardware system that grouped objects.

All of these implicit cooperative projects were successful, in that the robots were able to achieve the task desired in the absence of detailed communications. However, many of them succeeded by explicitly writing that task into the robots' software architecture.

3 Cooperative Insects

When describing cooperative systems, many sources make reference to the impressive complexity of social insects. The capacities of hive insects and the massive difference between the individuals and the constructs they build are more fully discussed elsewhere [Turner, 2011; Hölldobler and Wilson, 1995; Sudd, 1970; Theraulaz *et al.*, 2003] and will not be examined further here. However, the behavioural models developed to explain cooperative hive behaviour contain the basis for combining implicit, decentralised cooperation and general architectures.

Individual social insects are quite diminutive and have no central controller. Instead, they cooperate through the local interactions of individuals, [Theraulaz *et al.*, 2003] and an individual's perception of the environment. For example, Turner [2011] describes termite swarm cognition as being based on three major input elements: tactile input or sensor information (Turner's termites were blind), fluctuations in the local environment and a "rich medium of chemical communication between termites. . . "

Theraulaz *et al.* [2003] makes this more general, expressing social insect spatial patterns in terms of "template-based patterns" – where the building activities are controlled by the physical or chemical heterogeneity in the environment – and "stigmergy and self-organised patterns", where stigmergy controls the agent's actions. They also postulate that, in social insects, positive feedback results from social interactions – such as recruitment, imitation, etc., while negative feedback is caused by the environment.

In our architecture it is the stigmergic interactions and environmental feedback that are of most interest. Holland and Melhuish [1999] describe stigmergy as:

> ". . . a mechanism that allows an environment to structure itself through the activities of agents within the environment: the state of the environment, and the current distribution of agents within it, determines how the environment and distribution of agents will change in the future"

In other words, the agents gain the information they need from the current state of the environment; an environment that is being actively modified by other agents; they observe an environment that bears the marks of other agents' actions and base their own actions accordingly.

While insects do possess the ability to explicitly interact and communicate with nest-mates (chemical trails or markers, recruitment, etc.), we do not include such elements in our design. Such explicit avenues of communication suffer from the same problems as explicit communication among robots; we do not want to force the group into tight restrictions.

Instead, we focus on the observable environment to control behavioural selection. In this sense, our architecture could be considered an insect comprising only of Turner's first, sensory input element and Theraulaz's negative, stigmergic feedback. Probably the most succinct description of the insect we wish to copy is to be found in Gordon [1999]:

> "An ant is not very smart. It can't make complicated assessments. it probably can't remember anything for very long. Its behaviour is based on what it perceives in its immediate environment."

Theraulaz *et al.* [1998] discuss a response-threshold function for behavioural selection aimed at just such a creature. We use this model as the basis for our system of stigmergic observation based cooperation.

They base this model on three quantities: stimulus (s), threshold (θ) and probability (P), related by the following equation:

$$P_{i,j} = \frac{s_{i,j}^n}{s_{i,j}^n + \theta_{i,j}^n} \tag{1}$$

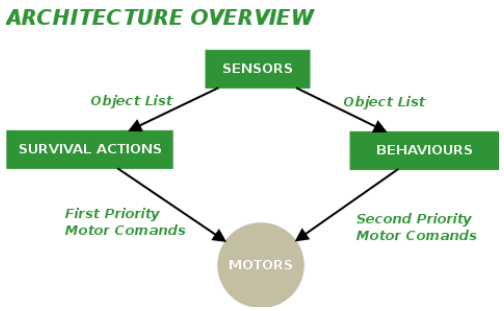

Figure 1: An overview of the architecture showing the three parallel spheres of operation and the Object List linking them.

Where:

i	refers to the agent i and j to behaviour j.
$P_{i,j}$	defines the probability of agent i enacting behaviour j
$s_{i,j}$	refers to the stimulus for that agent and behaviour. This is an increasing quantity that drives up $P_{i,j}$.
$\theta_{i,j}$	is the threshold for this agent and behaviour.
n	represents the degree of non-linearity desired in the relationship.

Brambilla *et al.* [2013] describes this relationship as a Probabilistic Finite State Machine (PFSM), as there are a finite number of behaviours available to each agent and some probability for the activation of each behaviour. According to Theraulaz *et al.* [1998] "individual i engaged in task j with probability" P. The threshold decreases with successful behaviour implementation and thus encourages successful behaviours to be re-implemented. It forms a primitive learning mechanism, leading ultimately to specialisation. This model reappears (with various differences) in a variety of sources [Schmickl and Crailsheim, 2008; Merkle and Middendorf, 2004]. Brambilla *et al.* [2013] also lists a number of other projects who have employed a PFSM of some form.

4 The Architecture Design

The central component of our architecture is the Object List. All sensor information is reduced to a list of recognised objects, recording their type and location relative to the observing agent. This is the only information about the surrounding world that is available to any component of the architecture.

In addition to the Object List, the architecture consists of three main areas (see Figure 1). All processes are intended to be executed in parallel.

1. **Sensor Interpreter,** responsible for filling the object list. It is here that object recognition takes place and sensor data is reduced to object type and location information. It was assumed at an early date that this would be a visually based sensor.
2. **Behaviours,** the basic building blocks of tasks. Behaviours follow fixed rules and are the users' interface into the architecture. They are intended to be created as "plug-ins" that could be pulled in or out without re-building the decision making process. Section 4.3 discusses the behaviours' construction and shows the two processes needed to drive them: a selection process (based on social insect models) and execution process. Behaviours are constructed from actions (see Section 4.2).
3. **Survival Process,** ensures that obstacle avoidance and other asynchronous survival actions can respond rapidly to changes in the environment without reference to the behaviours. At any time survival actions can take control of the motors based on the contents of the Object List. However behaviours may declare specific object types as exceptions.

4.1 Sensors and The Object List

While it would be very difficult to interpret the actions and intentions of another non-homogeneous robot successfully [Parker, 1998], it is relatively simple to recognise another agent. We proceeded from the assumption that all agents could be expected to bear an identifying mark (such as a coloured square or light). This was considered an acceptable requirement as such a marker could easily be placed on any machine.

We then took this idea further, assuming that all recognisable agents could be localised relative to the observing robot. This is another non-trivial problem, but modern computer vision is capable of this [Schmitt *et al.*, 2002]. Thus we could assume that other agents could be observed and their position known relative to the observer.

Taking this idea further again, we assumed that an observer robot would be able to identify and localise other objects besides its companion robots. Objects such as *target object*, *beacon object*, *obstacle object* and so on could be expected. This is a minor limitation but we considered it acceptable on the grounds that in a given environment useful objects will be fairly consistent across most conceivable tasks.

This creates a central object list in the local memory of each agent, comprising of whatever an agent can 'see' at that moment. This is the basis of our system and the only real-world information available to the other processes in the architecture.

4.2 Motor Actions

Although this is a behaviour based system, we wished to make the behaviours as formulaic as possible, reducing the necessary user input in behaviour creation with an eye for the eventual automation of behaviour construction. To make this possible, behaviours were tightly constrained and outsourced motor control loops to a predefined set of motor actions.

This was made possible by enforcing the concept of the Object List as the only legitimate source of information. If the only known information is the type and location of nearby objects, then the physical actions possible are sharply reduced. The programmer can only express movements in terms of object types and locations - either their presence or their absence.

On the surface, this appears to place us in the unenviable position of having to write an action for any conceivable combination of object, location and objective. Clearly, this is not feasible. However, if we assume that conceivable tasks are limited to manipulating objects in three-dimensional space (in any event, stigmergic cooperation requires that the agent move through and change objects in the environment [Holland and Melhuish, 1999]), all we are actually trying to do is move the agent to and from objects. Thus actions such as `move_to_object()` are universal for all object types.

The physical structure of the robot further limits the actions that can be performed. In this sense, the actions become motor driver functions that take the location of a given object (of any type) in space as input. A simple library of these can be written to cover the physical actions that the machine can perform.

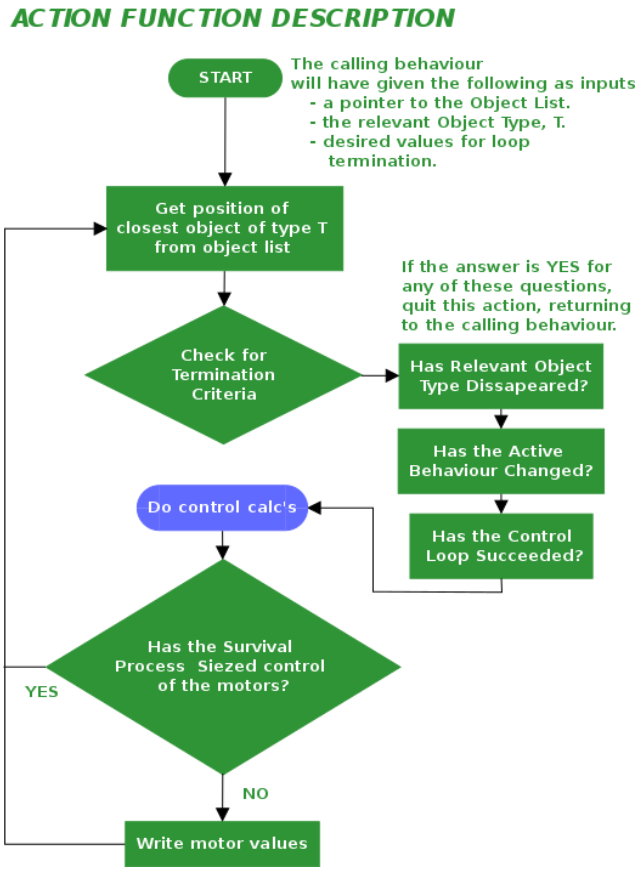

Figure 2: All motor action functions follow this basic template. The Green Blocks represent sections that are universal, only the rounded boxes are specific to each action. The purpose of action functions is to determine the current motor values for the effectors.

All actions ultimately comprise of a control loop which had a single objective describable in terms of one object type and its ultimate state. By convention they were written on the assumption that the closest instance of object type T was the desired specific object. Thus the input to all actions followed the same pattern: a pointer to the Object List, an integer object type label and a floating point number indicating the termination condition (eg: the minimum distance we wish to move to) if relevant. Figure 2 shows the action function format.

4.3 Behaviours

Behaviour based design often exhibits a distressing level of subjective intuition. Brambilla *et al.* [2013] noted that:

> "Unfortunately, in swarm robotics there are still no formal or precise ways to design individual level behaviours that produce the desired collective behaviour. The intuition of the human designer is still the main ingredient in the development of swarm robotics systems"

By combining the Object List with the Motor Actions to create Behaviours on a standard template, we aim to constrain this intuition within clear rules, to the point that behaviours are formulaic and require the absolute minimum of task-based information.

Behaviour are the users' interface into the architecture. Assuming that the recognisable objects and available actions are sufficient, modifying the behaviours allows simple and efficient means of addressing new tasks. Execution and selection are separate parallel processes. These components are illustrated in Figure 3.

In this architecture a behaviour is defined as a single goal, the accomplishment of which completes one aspect of the group's task. Behaviours are constructed from "action blocks", discrete units executed sequentially that define relevant object types and the actions to be taken. Behaviours are selected by an adaption of the insect model described in Section 3.

4.3.1 Behaviour Selection

Behaviours are selected based on a modification of Equation 1:

$$P_{i,j} = R_{i,j} \frac{s_{i,j}}{s_{i,j} + \theta_{i,j}} \tag{2}$$

Where:

- i refers to the agent i and j to behaviour j.
- P is the rank of this behaviour, the behaviour with the highest value of P is active.
- R is the relevance of this behaviour (see Section 4.3.2) and is either 1 or 0. This links the external information of the Object List with the internal behaviour selection process.
- s is the internal stimulus for this behaviour. This is a linear function of time and increases if the behaviour is inactive and decreases if the behaviour is active. All behaviours have the same rate of increase and decrease.
- θ is the internal threshold. This also evolves linearly with time, but in the reverse direction to s. Both stimulus and threshold are constantly moving for all behaviours, their direction is determined by the active or inactive status of a given behaviour.

Equation 4.3.1 provides the behaviour selection mechanism. This is continually calculated as behaviours are executed. If the active behaviour does not have the highest value of P, it is terminated and replaced with the new active behaviour. Once this occurs, the new active behaviour stimulus is set to 1 and all other stimuli are set to 0.

In contrast to Theraulaz *et al.* [1998] P has been simplified to become a first past the post rank.

The constants governing the increase of both θ and s were calculated by selecting initial conditions for stimulus and threshold (0 and 1) and the desired time required for an inactive behaviour to return a higher P value than the active behaviour (assuming in both cases that $R = 1$). We found that, if this time is 60 seconds the rate of change for s is $ds/dt \approx 0.0111$ and θ is $d\theta/dt \approx 0.0083$. The sign depends on whether the behaviour is active or inactive. In our simulation stimulus was initialised to 0 and threshold randomly allocated between 0 and 1.

However, not all behaviours will be relevant at all times; for instance, a robot cannot grasp an object of

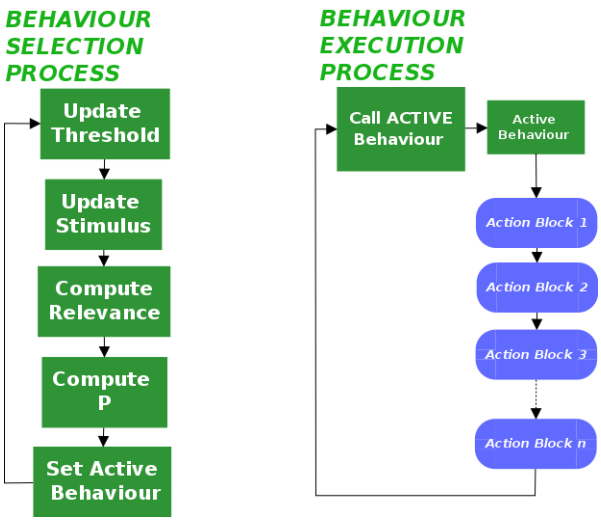

Figure 3: The behaviour selection and execution processes. Note that these operate in parallel, behaviour selection is continuously running. Actions are obliged to recognise when the active behaviour has changed and terminate. All elements of this diagram have access to the Object List. The rounded boxes represent the behaviour function elements. These can be set by the user. All else is intended to be part of the architecture and is not modified.

type T if no instances of T exist. Thus we created the concept of behavioural Relevance and included it into equation 1.

4.3.2 Relevance Calculation

Each behaviour requires certain object types to exist in order to be relevant. It may require these to exist in an *and* relationship (eg: $type_1$ and $type_2 \ldots$ and $type_n$) or the user may require a logic *or* relationship (eg: $type_1$ or $type_n$). Conversely, there may also be object types that must not exist for relevance. A search behaviour, for instance is not relevant unless the target object does not exist. Once again these non-existing objects may need to be arranged in and *and, or* relationship.

From this, we clearly have the *and, or, nand* and *nor* statements of classical logic, which we may view as *exist or_exist, not_exist* and *or_not_exist* requirements.

Thus each behaviour must specify what makes it relevant at any given time. There may be any number of object types defined in any relationship to each other, but experience has shown that behaviours work best when the fewest object types are needed, as there is no formal requirement preventing over-defined relevance.

When a behaviour is called, it uses functions provided by the main architecture to store its relevant object types in a common repository along with their logic requirements. The separate behaviour selection process continually computes the relevance of all the behaviours, along with P for all behaviours (see Figure 3). The current value of P is then multiplied by relevance.

Because the robots interact with their environment, the execution of behaviours will change the objects in view. What was relevant at the beginning of a behaviour will very likely be irrelevant by its end. As we did not wish to be restricted to one action per behaviour, we needed a mechanism that permitted relevance to be recomputed and updated as the behaviour advanced. To this end we created the concept of a Action Block.

4.3.3 Action Blocks

An Action Block is a single section of code that defines all information related to a single action and then executes that action. The full structure is described in Figure 4.

Behaviours are created by joining action blocks in sequence. This formalisation limits user input to defining the relevant object types, setting survival excepted types (see Section 4.4) and selecting the action itself. Each action block must check if the behaviour has been changed

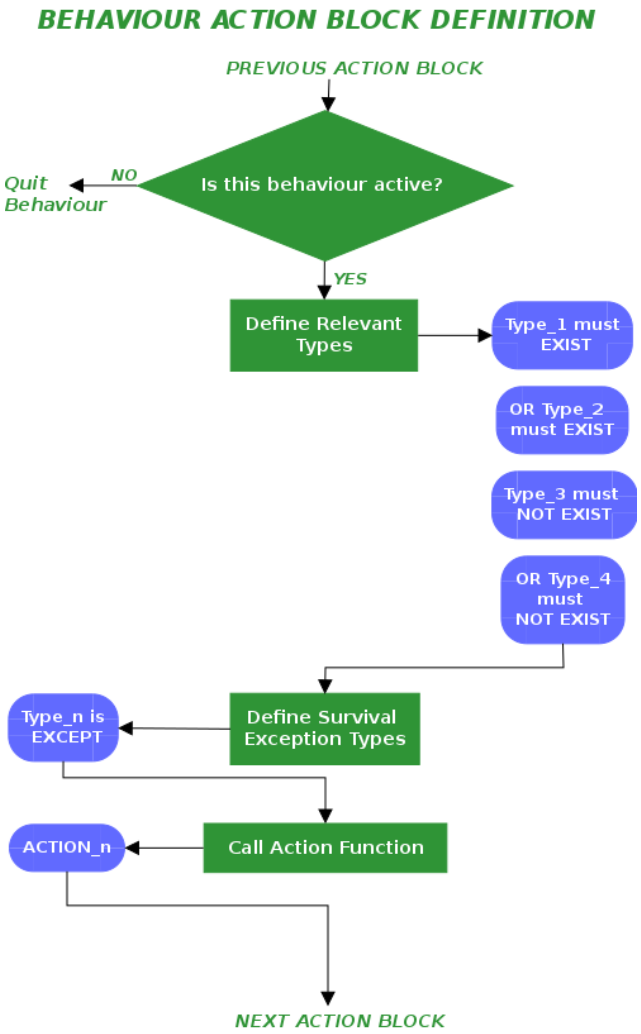

Figure 4: The action block format. The square boxes illustrate what is required, the rounded boxes what the user may modify for each action block. Note that it is not necessary to use all possible combinations of *exist* and *not_exist*. Any number of object types with any number of logical conditions may be used. Likewise, any number of exception types may be set.

before it begins. Action blocks do not contain explicit loops.

There is no formal limit to the number of action blocks in a behaviour. Behaviours are defined as a single goal of a broader task, but this goal may require several actions to complete. In practice, we found that a minimum of action blocks was desirable.

4.4 Survival Process

By constraining the behaviours as described above, we found it necessary to outsource survival actions to a separate process. This was considered a better solution than the alternative; forcing obstacle avoidance to be incorporated into behaviour or action formats. As it is conceivable that there would be times and places when survival actions would be unnecessary, behaviours can set object types that were to be ignored by the survival process. For example, a "pick up object" behaviour must be able to approach close enough to the object to acquire it. Therefore that behaviour would flag "object" as a type to be ignored by the survival process.

In our experiments we only used the one survival action: avoid. It simply took control of the motors and steered away from the closest object, if that object was not flagged an exception type and was within a minimum distance.

5 Simulation

To test these ideas we created a simulation using the ARGoS simulator (Autonomous Robots Go Swarming) [Pinciroli *et al.*, 2012]. The simulated robots are very simple agents, consisting (for our purposes) of two wheel motors arranged in a differential relationship, an omni-directional camera, a one degree of freedom gripper, proximity sensors and a coloured beacon (see Figure 5).

The test task was relatively simple: a number of agents were arranged randomly in a 2x2 meter square in the centre of a large field. In this square (also placed randomly) was a 'ball' (see Figure 5. The purpose of the task was to pass the ball from one agent to another. This very limited task enabled us to observe the interactions between agents. We wished to demonstrate that our architecture could fulfil the requirements of the task (pass the ball) without producing more data than could be

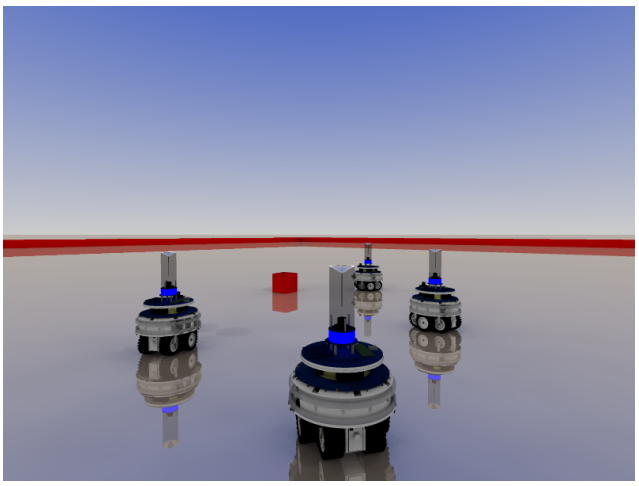

Figure 5: Still from a simulation run showing four robots and the 'ball'

easily analysed. We judged the success of the architecture by measuring how long each robot gripped the ball. This enabled us to assess the division of labour and the efficiency of the system. In an ideal scenario, the ball should have been gripped by any agent for nearly 100% of the simulation (which would indicate that it is being continually passed) and held for about the same time by all agents (indicating a equal division of labour).

Each robot was equipped with the following behaviours: `search_for_ball`, `search_for_agent`, `acquire_ball`, `move_ball_2_agent` and `receive_ball`.

The ball was identifiable by a red light. The robots were identified by a blue light. The ARGoS two-dimensional physics engine was used and all objects distributed using a uniform, Gaussian distribution. The computer's system clock was used as the random seed, ensuring that no two simulations employed the same pseudo-random number sequence. All simulations were terminated at 600 seconds.

The robots were permitted to roam around the field at will, but could only see two meters in any direction. No artificial noise was added to either the sensors or the motors, both of these being considered 'perfect'.

6 Results

We conducted a number of simulations of the type described in Section 5. We investigated the length of time the ball was gripped by a robot and how this was effected by the number of robots in the simulation.

Number of Robots per Simulation	**Average Grip Duration (%)**	**Standard Deviation**
3	83.45	15.84
4	69.62	9.43
5	57.91	13.14
8	34.15	9.95

Table 1: The average time the ball was gripped by the robots as a percentage of the 600 second simulation runs – this is the percentage time the ball was in 'play'. There were four different groups sizes and 10 simulations per group size.

Initially the task was designed for four machines. We ran simulations with groups of three, four, five and eight robots. For each group size we ran ten simulations of 600 seconds duration. Table 1 shows the results of all simulations. Figure 6 shows the results from the ten four robot simulations.

7 Discussion

Figure 6 shows that each robot in the four robot group spends approximately the same amount of time holding the ball. This illustrates that there is a roughly even distribution of labour. Interestingly, we found that this was similar for other group sizes, although Table 1 shows that the total gripped time decreases with group size. Thus we would argue that, for these group sizes and task, there is little evidence of specialisation. Possibly this would appear if the groups were larger and distributed over a wider area or the task was more complex. In the above tests the robots could see a large part of the arena, which probably retarded specialisation as the object lists would be similar for all agents.

The fairly equal distribution of labour indicates that cooperation is occurring. In addition to this, the total gripped time is not insignificant, especially for smaller groups. Thus this group is not wasting time. However, as the total gripped time decreases with group size (Table 1) larger numbers of robots increasingly get in each-others way. However, we believe that these results demonstrates that the architecture functioned as a co-operative system; work was being done for a significant

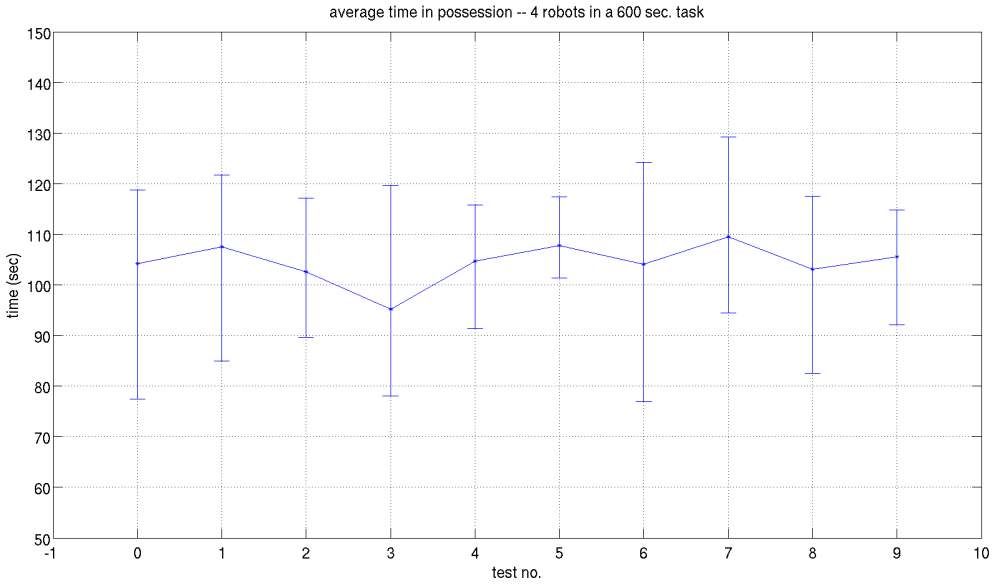

Figure 6: The results of 10 simulations for a four robot group. This plot shows the average time each robot in the group has the ball (seconds) in each 600 second simulation. The maximum and minimum times for each simulation are also shown.

percentage of the time and division of labour was reasonably equal.

What would be the practical applications of this system? Brambilla *et al.* [2013] list a number of applications for swarm and cooperative robots:

> "...including exploration, surveillance, search and rescue, humanitarian de-mining, intrusion tracking, cleaning, inspection and transportation of large objects"

This is what could be described as the usual list, repeated with minor variations in most cooperative robot sources. While we do not have any particular task in mind for this architecture, its internal constraints and observations of the simulation suggest some guidelines for practical applications.

The most important of these is the absence of the definite article. As mentioned in Section 1, this architecture was not designed with detailed instructions in mind. This is realised by the description of behaviour relevance by object type, rather than specific objects and the convention that all actions relate to the closest interest of type T. This lack of precision means that actions that may seem vital to a human observer will be ignored by a robot, thus they may not respond in a fashion that a human may consider logical (although it is logical from the robot's limited point of view). In all tasks what an individual robot can achieve is limited by its restricted perspective.

Because of this we consider that the most applicable tasks are ones that enable the relatively simplistic agents to operate at their own pace. Tasks that require continuous execution, like cleaning or digging would be very applicable. Ultimately we envisage a central machine with a broader understanding of the environment sending out teams of autonomous agents to solve tasks without direct oversight. Our architecture would be especially suited to a colony approach. A colony could be established with any number of blank worker robots who could have new behaviours installed at run-time in response to new situations and then left to solve them on their own.

We believe that such colonies would be of most benefit in very hostile environments where direct human oversight is impossible, such as radiation hazard zones, re-

mote locations, deep sea, or space environments.

8 Future Work

In this paper we have introduced the basic concepts of our architecture and conducted proof of concept tests. In the near future we hope to expand the simulation to a more complex task. Simple 'construction', 'sorting' and 'cleaning' tasks are being considered.

We are also in the process of implementing the 'passing' task in hardware. At the time of writing, the physical robots are slightly simpler than their simulated counterparts (a non-articulated scoop rather than a gripper and a forward facing camera), but will provide insight into the real world problems of the architecture. We anticipate that sensing will be noisier and less reliable than in the simulation. We have added a 'confidence' value on each detected object and changed Relevance from a Boolean to floating point value to account for this. We hope to have the hardware version operational in the near future.

If successful, the hardware implementation will illustrate that the architecture is sufficiently robust to function in the real world. The more complex simulations will demonstrate that the architecture is capable of producing cooperation on practical tasks.

9 Conclusion

We have presented an new architecture for implicit, decentralised robotic cooperation. This architecture was intended from its inception to be separated as much as possible from task specific information and not to require strong inter-agent homogeneity as would be required for explicit communications. It is based on previous behavioural robotic projects and biological models for hive insect societies.

We have demonstrated through a simple simulated task that it is capable of successful cooperation and a consistent division of labour. We hope to shortly expand this demonstration to more complex tasks and real world operation.

We believe that this design addresses at least some of the limitations of previous robotic groups and could be applied to a number of real-world problems, ultimately creating autonomous robotic colonies.

References

Noa Agmon and Peter Stone. Leading multiple ad hoc teammates in joint action settings. In *Interactive Decision Theory and Game Theory*, 2011.

R. C. Arkin. *Behavior-Based Robotics*. MIT Press, 1998.

George A. Bekey, Robert Ambrose, Kumar Vijay, David Lavery, Arthur Sanderson, Brian Wilcox, Junku Yuh, and Yuan Zhery. *Robotics, State of the Art and Future Chalanges*. Imperial College Press, 2011.

Manuele Brambilla, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41, 2013.

Rodney A. Brooks. *Robot: The Future of Flesh and Machines*. Penguin Books, 2002.

Cristiano Castelfranchi. Silent agents: From observation to tacit communication. In Jaime Sichman, Helder Coelho, and Solange Rezende, editors, *Advances in Artificial Intelligence - IBERAMIA-SBIA 2006*, volume 4140 of *Lecture Notes in Computer Science*, pages 98–107. Springer Berlin / Heidelberg, 2006. 10.1007/11874850_14.

Erol Şahin. Swarm robotics: From sources of inspiration to domains of application. In Erol Şahin and William Spears, editors, *Swarm Robotics*, volume 3342 of *Lecture Notes in Computer Science*, pages 10–20. Springer Berlin / Heidelberg, 2005. 10.1007/978-3-540-30552-1_2.

Deborah M. Gordon. *Ants at Work, How an Insect Society Is Organised*. Simon & Schuster Inc., 1999.

Owen Holland and Chris Melhuish. Stigmergy, self-organization, and sorting in collective robotics. *Artif. Life*, 5(2):173–202, April 1999.

Bert Hölldobler and Edward O. Wilson. *Journey to the Ants: a story of scientific exploration*. Harvard University Press, 1995.

T. Huntsberger, P. Pirjanian, A. Trebi-Ollennu, H. Das Nayar, H. Aghazarian, A.J. Ganino, M. Garrett, S.S. Joshi, and P.S. Schenker. Campout: a control architecture for tightly coupled coordination of multirobot systems for planetary surface exploration. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 33(5):550 – 559, sept. 2003.

AukeJan Ijspeert, Alcherio Martinoli, Aude Billard, and LucaMaria Gambardella. Collaboration through the exploitation of local interactions in autonomous collective robotics: The stick pulling experiment. *Autonomous Robots*, 11:149–171, 2001. 10.1023/A:1011227210047.

Serge Kernbach, editor. *Handbook of Collective Robotics: Fundamentals and Challanges*. CRC Press Taylor & Francis Group, 2013.

Jelle R. Kok, Matthijs T.J. Spaan, and Nikos Vlassis. Non-communicative multi-robot coordination in dynamic environments. *Robotics and Autonomous Systems*, 50(23):99 – 114, 2005. Multi-Robots in Dynamic Environments.

Maja J. Matarić. Behaviour-based control: examples from navigation, learning, and group behaviour. *Journal of Experimental & Theoretical Artificial Intelligence*, 9(2-3):323–336, 1997.

Daniel Merkle and Martin Middendorf. Dynamic polyethism and competition for tasks in threshold reinforcement models of social insects. *Adaptive Behavior*, 12(3-4):251–262, 2004.

Eduardo Montijano, Johan Thunberg, Xiaoming Hu, and Carlos Sagues. Multi-robot distributed visual consensus using epipoles. In *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*, pages 2750 –2755, dec. 2011.

Enrico Pagello, Antonio DAngelo, Federico Montesello, Francesco Garelli, and Carlo Ferrari. Cooperative behaviors in multi-robot systems through implicit communication. *Robotics and Autonomous Systems*, 29(1):65 – 77, 1999.

L.E. Parker. Alliance: an architecture for fault tolerant multirobot cooperation. *Robotics and Automation, IEEE Transactions on*, 14(2):220 –240, apr 1998.

Tuan A Phan and R Andrew Russell. A swarm robot methodology for collaborative manipulation of non-identical objects. *The International Journal of Robotics Research*, 31(1):101–122, 2012.

Carlo Pinciroli, Vito Trianni, Rehan O'Grady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni Di Caro, Frederick Ducatelle, Mauro Birattari, Luca Maria Gambardella, and Marco Dorigo. ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, 6(4):271–295, 2012.

T. Schmickl and K. Crailsheim. Taskselsim: a model of the self-organization of the division of labour in honeybees. *Mathematical and Computer Modelling of Dynamical Systems*, 14(2):101–125, 2008.

T. Schmitt, R. Hanek, M. Beetz, S. Buck, and B. Radig. Cooperative probabilistic state estimation for vision-based autonomous mobile robots. *Robotics and Automation, IEEE Transactions on*, 18(5):670 – 684, oct 2002.

J.N. Schwertfeger and O.C. Jenkins. Multi-robot belief propagation for distributed robot allocation. In *Development and Learning, 2007. ICDL 2007. IEEE 6th International Conference on*, pages 193 –198, july 2007.

John H. Sudd. *An Introduction to the Behaviour of Ants*. Edward Arnold, London, 1970.

G. Theraulaz, E. Bonabeau, and J-N. Denuebourg. Response threshold reinforcements and division of labour in insect societies. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 265(1393):327–332, 1998.

Guy Theraulaz, Jacques Gautrais, Scott Camazine, and Jean-Louis Deneubourg. The formation of spatial patterns in social insects: from simple behaviours to complex structures. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 361(1807):1263–1282, 2003.

J. Turner. Termites as models of swarm cognition. *Swarm Intelligence*, 5:19–43, 2011. 10.1007/s11721-010-0049-1.

Appendix B

Implementation

The Architecture was implemented upon two swarms, real and simulated robots. The simulation was relatively simple – with "perfect" vision, low-level sensing and effector control. The real-world robots were much more troublesome to implement. The original design was over-optimistic in its simplicity, necessitating several generations of re-design.

It should be stressed that the real-world machines are not intended to be perfect robots. They were designed to be cheap, simple swarm agents. They had to be implemented by one person with restricted time on a very limited budget. Although "better" swarm machines could be built, these machines are still swarm agents in the sense discussed in Chapters 1 and 2.

B.1 Real-World Robots

B.1.1 Evolution

To maintain robot simplicity and reduce cost and implementation time, the original design used the camera as the robots' only sensor. This was considered the minimum hardware needed to identify object types and localise objects.

In practice, the vision system was not sufficiently reliable to be the only sensor. The field of view was too narrow and the result too prone to false identification to be used without any additional support.

To overcome this, a "gripper" sensor was added to identify captured objects and touch sensors were attached to the rim of the robot to identify proximity-objects.

The original robots possessed only a static "scoop" as the gripper. It was felt that an articulated pincer would simply add time and expense to the development process without materially adding to the project. A static scoop could still perform the "grip" and "drop" actions simply by moving the robot's wheels.

Again, this proved too simple, as avoidance caused the robots to inadvertently drop captured objects. This was overcome by adding a simple "hook" powered by a small servo to the scoop. This one-degree-of-freedom "finger" secured captured target objects.

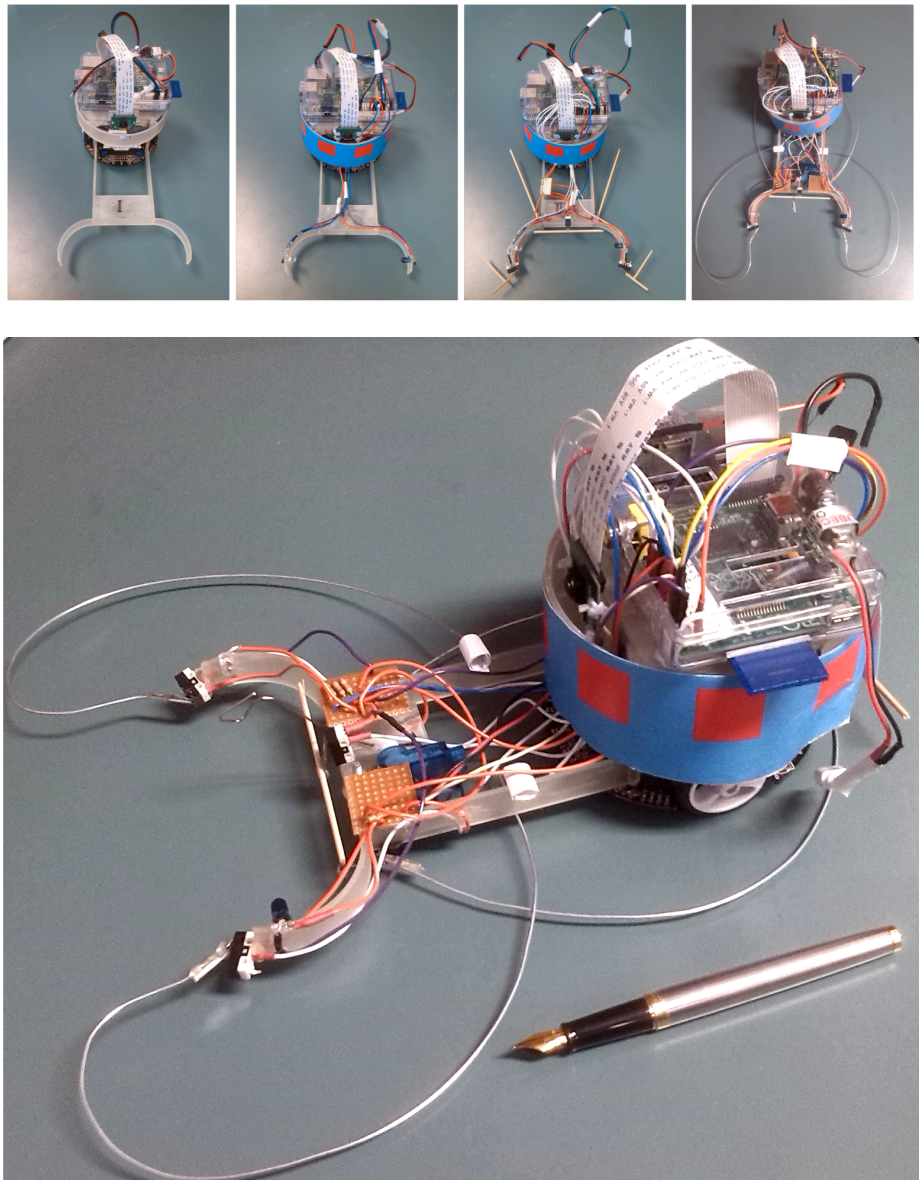

Figure B.1. Top: The steady evolution of the robotic hardware. Left to right: the original, camera-only machine, the addition of the infrared capture sensor, the first touch sensor design and the final wire touch whiskers and powered hook. Bottom: the final robot.

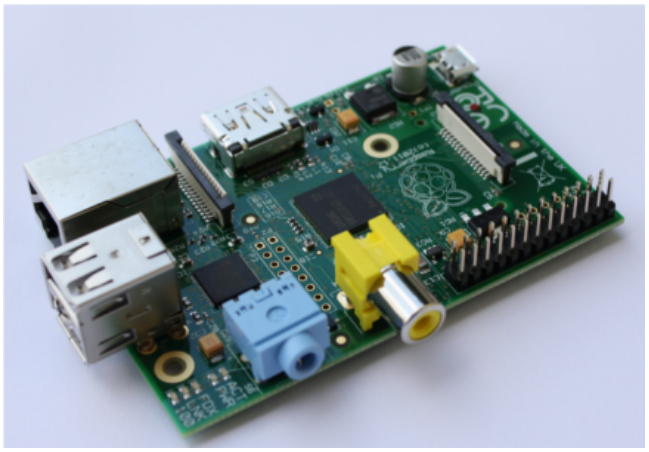

Figure B.2. The Raspberry Pi model B (revision 2) single board computer – the primary computer in all physical agents. From the Raspberry Pi Foundation.

This evolution process was achieved through trial and error and emphasised that, while the four actions of The Architecture allow for simple agents, the robots must be able to perform each action robustly.

The sequence of robot development is illustrated in Figure B.1.

B.1.2 Computer

The primary computer was a Raspberry Pi model B (revision 2) single board computer (see Figure B.2). While use was made of the GPIO pins for capture detection and communication to the wheels, no modifications were made to the board itself.

The Operating System was Raspian, (Debian Wheezy Linux optimised for use on this platform). The only noteworthy modifications were to the `rc.local` script to obtain boot-time operation of the architecture and an overclock from 700 to 900Mhz.

More information on Raspian is available at: `raspberrypi.org`

The computer was powered by a 6V, 2000mAh rechargeable NiMH battery from Eneloop, running through a 5V 3A BEC Step-Down Voltage Regulator by Pololu Robotics and Electronics (see `pololu.com/product/2177`) for added protection.

B.1.3 Low-Level Sensors

Touch

The Raspberry Pi GPIO pins provide the interface for all sensors bar the camera (which has its own dedicated CSI port). The connections for the sensors is given in Table B.1.

The Touch sensors are simply a sprung contact switch (see Figure B.4) with an extended contact arm (a wire loop).

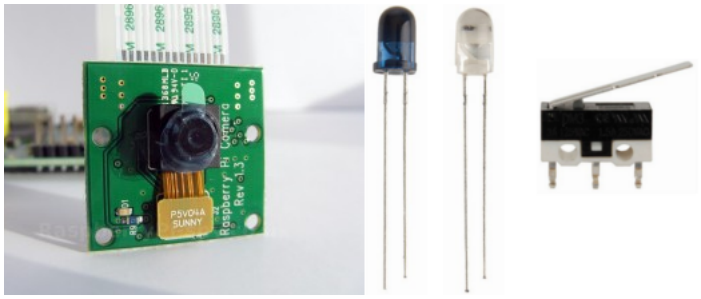

Figure B.3. Left: The Raspberry pi camera used as the robot's primary sensor. From the Raspberry Pi Foundation. Right: The low-level sensor electronics. Left to right: the scoop gate infrared diode, the infrared photo-transistor and the touch sensor contact switch (not to scale). See `jaycar.com.au` catalogue numbers ZD-1945, ZD-1950 and SM1036 respectively.

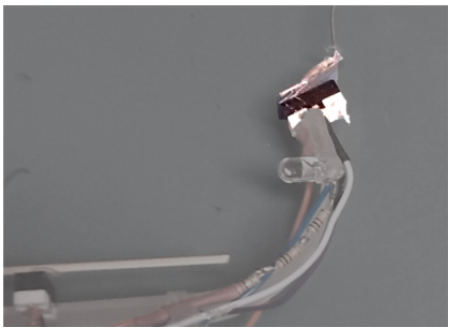

Figure B.4. The touch sensor switch, attached to right scoop arm.

Sensor	BCM Pin No.
touch: scoop, right	7
touch: scoop, centre	25
touch: scoop, left	8
touch: flank, right	24
touch: flank, left	22
touch: rear	17
infrared gate	23

Table B.1. The low-level sensor configurations. The touch sensors are all contact micro switches. The pin numbers denote the Broadcom pin numbers for the Raspberry Pi B revision 2. Note that the wiringPi library used to interpret these pins uses a different numbering scheme.

The switches can only be triggered if pressed, not pulled. More advanced designs would have contact-less infrared or ultrasonic proximity sensors. These were too deemed expensive to implement in this project (they would have been required in large numbers and demanded more complex implementation), however would have been better in hindsight, as the contact arms were prone to catching in other robots.

The six touch sensors were attached in the following order:

1. One to each scoop wing, facing forward to detect collisions from approximately 10 to 2 o-clock.

2. One in the scoop itself to avoid obstacles becoming trapped in the scoop (the target objects are made from paper and as such are too light to overcome the spring).

3. One on each flank, just behind the scoop to cover the sides.

4. One in the rear.

Capture

Originally, it was hoped that the camera would be able to recognise captured objects without the need for another sensor, any object within view closer than x centimetres would be seen as "captured".

In the event, this proved impractical, as visual recognition was poor that close to the camera and conflicting geometries (the need to recognise objects both close and far away) were irreconcilable. As the ability to discriminate between the target and captured object types was essential, the robots were fitted with an infrared "gate" across the mouth of their scoops (compare Figure B.1 top, left and second from left). A break in the gate (logic low at the phototransistor emitter) indicated the presence of a captured object.

The circuit necessary to achieve this is depicted in Figure B.5. The voltage necessary to run the gate was sourced from the robots' master computers' GPIO pins, while the signal (V_{out}) was connected to digital pin on the same computer.

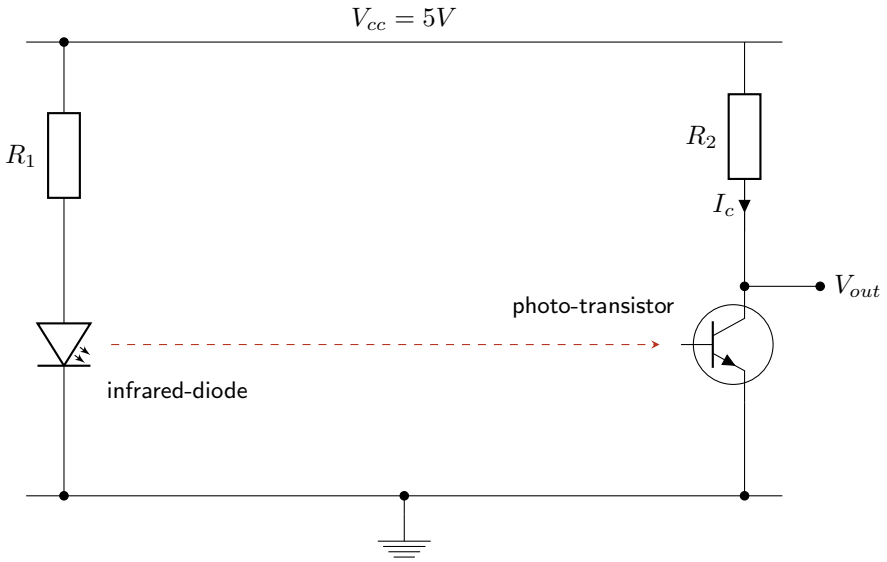

Figure B.5. The infrared circuit design. Left, the photo-transistor, right the infrared LED. The diodes were mounted on the left of the robots' scoop, the photo-transistors on the right.

The circuit is configured so V_{out} has a potential of 0V (or below the computer's "logic low") in "full light" (no object). When this occurs, the voltage is given by the following equation:

$$V_{out} = 0 = V_{cc} - I_c R_2 \tag{B.1}$$

Given that $V_{cc} = 3.3V$ (The robot computer's operating voltage – see Appendix B, and that measured values I_c in full "light" were around 1.5mA, the value of R_2 can be computed thus:

$$\begin{aligned} R_2 &= \frac{V_{cc}}{I_c} \\ R_2 &= \frac{3.3}{1.5 \times 10^{-}3} \\ R_2 &= 2.2\text{k}\Omega \end{aligned} \tag{B.2}$$

Thus a voltage change at V_{out} from low to high indicated the presence of a captured object. See Figures B.5 and B.1 (second from left) for implementation.

The infrared diode is run from the same 3.3V power rail as the current draw required to run the diode is very small (an R_1 value of 220Ω drew only approximately 11mA and produced more than enough "light").

As we shall see, the presence of captured objects was used to obtain quantitative results on Architecture viability (see Section 4.2). As the infrared gate was unable to distinguish between a captured target object and a blockage caused by any other factor, collisions between robots occur and unquestionably add false positives to the results.

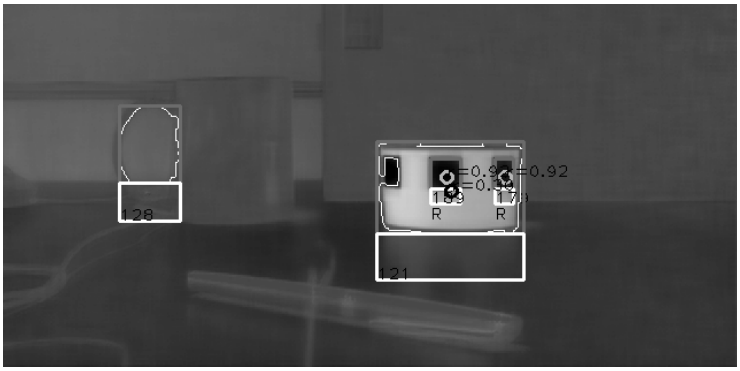

Figure B.6. The view from the robot. This is a still from the visual feed from a single robot, showing object detection in a complex scene (this is the V channel, the image contrast has been adjusted for ease of viewing). The robot is sitting on a desktop looking at a black computer tower (background, right) with objects on the wall behind it (background, left). Also present in the image are a coffee mug (centre left), a target object (centre right) and a pen (foreground). The robot has identified the handle of the coffee mug, the entirety of the paper target object and two of the object markers as potential objects. The markers are the only positive matches.

This is a problem, but it is not considered significant. The duration of these episodes is small overall, nor was absolute accuracy in the results considered necessary for proof-of-concept verification.

B.1.4 Vision

Overview

The full vision process is as follows (the Open Computer Vision library – `opencv.org` – was used throughout):

- The image was thresholded in the V channel (of the input YUV422 image). As the identifying square always contrasted well with the object main colour, this separated the square from the background. An "erode" morphology was used to remove small noise blobs.

- The resultant binary image converted to contours and the shapes processed using Hu moment analysis footnoteSee Hu (1962) or Bradski & Kaehler (2008) for more on this method. and by computing the hull defect sum for each object[1].

  Two methods of shape detection were not strictly necessary, but were a relic from earlier, purely shape-based sensor interpreters. In the event,

[1]This was rendered scale invariant by dividing by the length of the contour, thus allowing the same object to have the same hull-defect sum at various distances. The Hu moment analysis is already scale-invariant (Hu 1962, Bradski & Kaehler 2008).

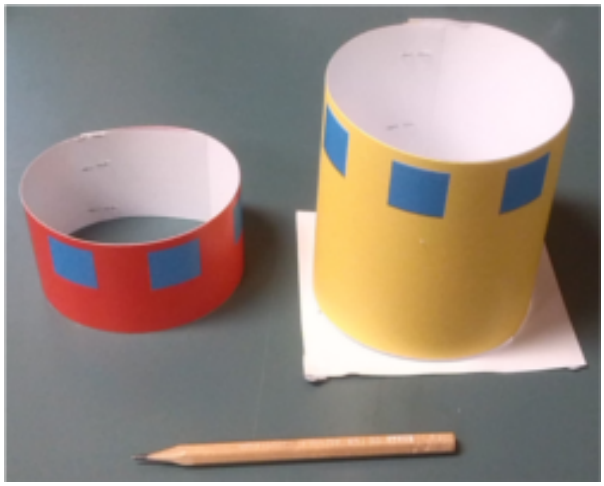

Figure B.7. The final target objects, Left: the first target object used in all tasks, right: the second target object, used in the grouping2 task.

removal of one method did not appreciably increase the frame-rate, while decreasing reliability.

- Object type was identified using the background colour of the object. A histogram of a rectangular area under the marker square was taken, the peak of this histogram (an integer index from 0 to 255) was used to determine the colour by comparing it to pre-determined thresholds for each colour.
- Range and bearing were then computed (see above).
- Objects were

Objects, Colours and Blur

In the real-world experiments, objects were paper cylinders with square markers of known size and colour identifying them as object types (see Figure B.7.

The square marker distinguished the objects from the background, while the background colour of object identified the type. The largest problem with vision was the "motion-blur" caused by the panning of the camera as the robots turned. If the robot turns rapidly, the camera image becomes horizontally blurred, objects loose clear shape definition and colours smooth out. To overcome this, the robots' were restricted to a low turning speed and the object marker set as a square, which is a simple shape and resistant to horizontal blur.

Object type classification was performed using the object background colour *under* the square marker. This is a patch of colour that did not noticeably change colour as the camera panned. The uninterrupted horizontal band is too large for the surrounding background colour to appreciably "bleed" into the band[2].

[2]The author would like to thank Dr. Richard Landers for pointing this out.

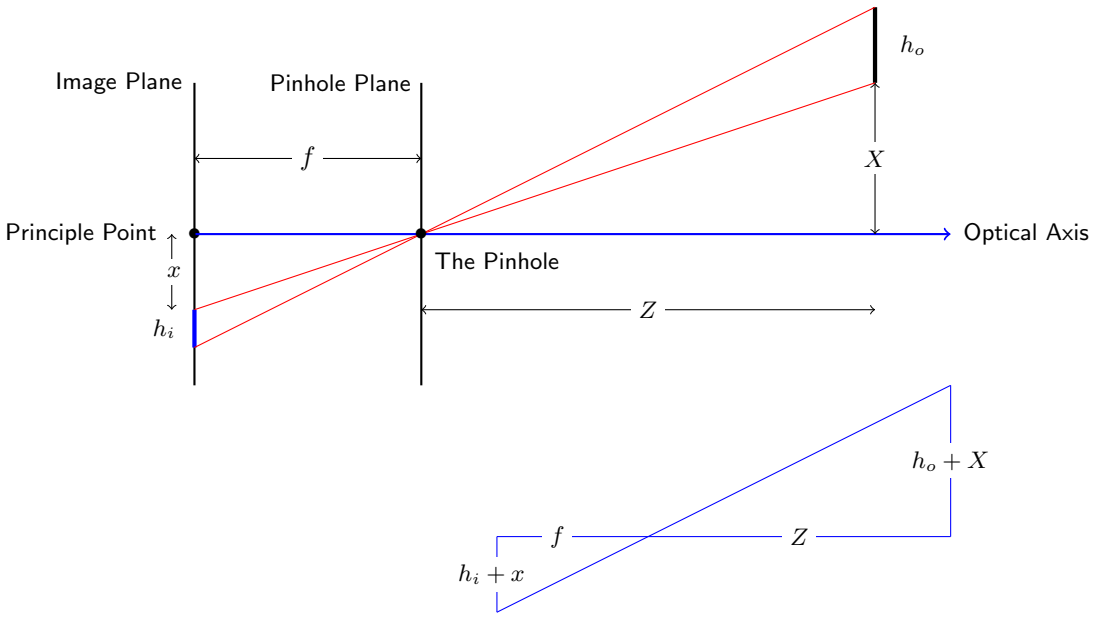

Figure B.8. Top: Pinhole camera model, side view: illustrating of the relationship between object height (h_o), image height (h_i), focal length (f) and depth (Z). Bottom: the similar triangles that may be formed from this model.

Originally, only the V channel was used (the U and V channels were found to be far more resistant to changes based upon light – see Florczyk (2005)). This was insufficient to reliably identify three distinct objects for the grouping2 task and was supplemented with the U channel. The red target-object-1 colour and the blue agent-object colour (see Figures B.1 and B.7) were chosen for their dissimilarity in the V channel. The yellow object-2 colour was chosen for its dissimilarity to the first two in the U channel.

Range and Bearing

Once the object was identified and classified, the known real-world size of the marker square, the known camera focal-length, known pixel-size and the pinhole camera model were exploited to find its (very rough) position relative to the robot (see Figures B.8 and B.9).

This is the simplest and most ideal form of a camera; a lens-less camera represented by a hole of zero area bored in a card. The card is held at distance f from the imaging surface (the digital imager) and a beam of light travels from the real-world object – which is at distance Z from the pinhole – through the pinhole and forms an inverted image on the image plane (Bradski & Kaehler 2008, Hecht & Zjac 1974).

It is important to note that Z is the distance from the pinhole to the object along the optical axis of the camera – in other words, it is the Z coordinate of the object in three dimensional space, with a coordinate system centred at the

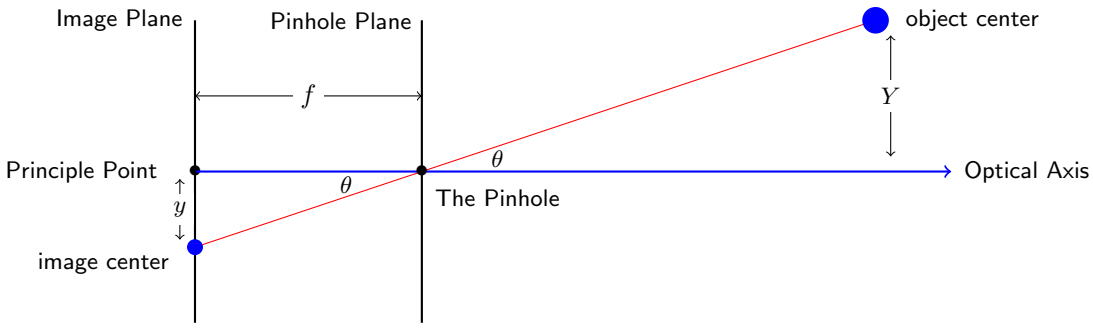

Figure B.9. Pinhole camera model, top view: by using similar triangles, and knowing the value of f and y, the bearing of any real-world object θ may be computed.

pinhole – not the direct distance from the object to the pinhole.

Straight lines drawn from the top and bottom of an object passing through the pinhole to the image plane produce two similar triangles which can be made into right angle triangles as demonstrated in the lower right corner of Figure B.8. Similar triangles enables the following equality to be made:

$$\begin{aligned} \frac{h_i + x}{f} &= \frac{h_o + X}{Z} \\ \therefore Z &= \frac{f(h_o + X)}{h_i + x} \end{aligned} \tag{B.3}$$

Again, using similar triangles, we can see that:

$$\begin{aligned} \frac{x}{f} &= \frac{X}{Z} \\ \therefore X &= \frac{Zx}{f} \end{aligned} \tag{B.4}$$

By substituting equation B.4 into B.3 for X, we can show that Z can be found from f, h_o and h_i regardless of the values of X or x:

$$\begin{aligned} Z &= f\frac{h_o + \frac{Zx}{f}}{h_i + x} \\ Z &= f\frac{h_o + Zx}{h_i + x} \\ Z(h_i + x) - Zx &= fh_o \\ Z(h_i + x - x) &= fh_o \\ Zh_i &= fh_o \end{aligned}$$

$$Z = \frac{fh_o}{h_i} \tag{B.5}$$

Once the depth is known, it is necessary to compute the corresponding bearing. This can be achieved by using the y coordinate of an object as demonstrated in Figure B.9.

$$\begin{aligned} \tan(\theta) &= \frac{y}{f} \\ \therefore \theta &= \arctan(\frac{y}{f}) \end{aligned} \tag{B.6}$$

Once both depth and bearing are known, they can be combined to give range R as follows:

$$R = \frac{Z}{cos(\theta)} \tag{B.7}$$

Thus on a flat plane an object of known dimensions can be localised relative to the camera in terms of range, R and bearing, θ. This method was suggested by the work of Schmitt, Hanek, Beetz, Buck & Radig (2002).

To obtain the camera focal-lengths the OpenCV camera calibration functions were used. See Bradski & Kaehler (2008) for details and attached DVD for implementation code. In practice this method was very crude.

Tracking

Objects were tracked from frame to frame to enable both confidence calculation and limited "projection" of position once the object left the camera's field of view.

For tracking, two lists of objects were maintained, one from the past (frame_{k-1}) and one from the current-frame (frame_k). Objects from the old frame were matched to objects in the new frame. This order was simply the easiest to code. If a successful match occurred, the confidence entry in the new list was incremented and the old position entries filled with the position of the old object.

If no match was found, the unmatched object was projected based on its last tracked position and its confidence decremented. Projection was based on the difference between position at frame_{k-1} and frame_{k-2}, both of which were recorded. This difference was added to the position at frame_{k-1} to produce a rough guess at frame_k position. This assumed that movement from frame to frame is both smooth and small.

If no last position existed, then it would be "projected" in place. Objects with a zero or negative confidence were purged. All remaining matched or projected objects were then entered in the global list.

If a new object remained un-matched at the end of this process, it was considered a new object and entered in the global list with an initialised confidence and no history

In practice it was found that objects could not be tracked for too long, or else the movement of the robot rendered their projected positions meaningless. This was especially true when the robot turned as the projection was based exclusively on the object's previous evolution in the image. If the robot changed

speed after the object left the image then the projection assumptions were undermined. Two to three seconds were found to be the working maximums for this crude system.

Confidence Calculation

The confidence of objects was based upon the quality of recognition and the duration of observation. The better the shape match (to a perfect square, see above) and the longer the object had been tracked, the higher the confidence.

To pass the shape match test a potential object had to return Hu moment and hull error values of less than hu_{max} and $hull_{max}$ respectively. This meant that, if valid Hu and hull error values were to be divided by their maximum acceptance thresholds, the results would be floating point numbers between zero (perfect match) and one (at threshold) and capped between 0 and 1. Using this property, a match criterion m was defined as a dimensionless number between zero and one and calculated by taking the product of one minus the two divisions, as shown in Equation B.8.

This process meant that an m value of 0 corresponded to a very bad match, while $m = 1$ signified a very good match. Thus an assessment of object match was arrived at – see Equation B.8.

$$\begin{aligned} hu_n &= \frac{hu}{hu_{max}} \\ hull_n &= \frac{hull}{hull_{max}} \\ m &= (1 - hu_n) \times (1 - hull_n) \end{aligned} \tag{B.8}$$

Once m had been successfully calculated it was stored in the object list along with the other object data. If that object was successfully tracked, it was retrieved and used to calculate confidence.

The function combining object duration and m needed to return high confidences for objects with a good match ($m \to 1$) and long tracking duration ($t \to t_{max}$). Thus if a very good match appeared suddenly, it was rewarded with high confidence, regardless of its duration, permitting the robot to respond quickly to good objects.

For the sake of simplicity this was achieved by defining flat plane in time, match, confidence space from the following points:

$$\begin{aligned} P &= [0, m_{max}, c_{max}] \\ Q &= [t_{max}, 0, c_{max}] \\ R &= [0, 0, 0] \end{aligned}$$

Where:

c_{max}	is the maximum possible confidence value,
m_{max}	is the maximum possible match value,
t_{max}	is the time desired for an object confidence to climb from zero to c_{max} (in seconds) and
t	is the duration time of this object (in seconds)

Using these three points, two vectors may be constructed defining the plane, thus:

$$\begin{aligned} \vec{RP} &= [t_{max}, 0, c_{max}] \\ \vec{RQ} &= [0, m_{max}, c_{max}] \end{aligned}$$

Using the property that the cross product of these two vectors gives an orthogonal vector, we may compute the confidence plane's normal:

$$\begin{aligned} \vec{n} &= \begin{vmatrix} \vec{\imath} & \vec{\jmath} & \vec{k} \\ t_{max} & 0 & c_{max} \\ 0 & m_{max} & c_{max} \end{vmatrix} \\ \vec{n} &= -\vec{\imath}(c_{max}m_{max}) - \vec{\jmath}(t_{max}c_{max}) + \vec{k}(t_{max}m_{max}) \\ \vec{n} &= [(-c_{max}m_{max}), (-t_{max}c_{max}), (t_{max}m_{max})] \end{aligned}$$

Using the equation of a plane: $\vec{n} \cdot (x, y, z) = 0$ it is possible to derive an equation for confidence (c):

$$\begin{aligned} 0 &= \vec{n} \cdot (x, y, z) \\ 0 &= x(-c_{max}m_{max}) - y(t_{max}c_{max}) + z(t_{max}m_{max}) \end{aligned}$$

Which can be re-arranged to give

$$c = \frac{(c_{max} \times m_{max} \times t) + (t_{max} \times c_{max} \times m)}{t_{max} \times m_{max}} \tag{B.9}$$

Where, c is the confidence value.

Which, as c_{max} and m_{max} are both equal to one, becomes:

$$c = \frac{t + (t_{max} \times m)}{t_{max}} \tag{B.10}$$

Using Equation B.10, the confidence for any tracked object may be found, provided m and t are known. In all cases confidence was capped at $c_{max} = 1$ regardless of the output of Equation B.10.

Once tracking had failed and object projection begun, confidence was permitted to decay linearly at a fixed rate per unit time, allowing for a few seconds of extended existence.

Vision Evolution

The visual system was the most difficult and time-consuming system to develop. It went through several iterations and revisions:

- *Marker placement* – Originally, the marker would have been a coloured ball held above the robot. This scheme did not take the cameras' limited field of view into account and could only recognise objects situated a long distance from the observing agent.

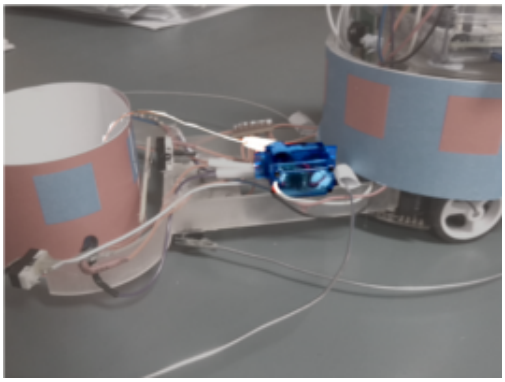

Figure B.10. The micro-servo motor powered hook being used to hold an object in the gripper scoop.

The second iteration used a rectangular colour band of known width around the "waist" of the robot. This approach was discontinued due to problems with occlusion – the bands were easily obscured by other robots, two robots in close proximity became one very large robot.

- *Different colour systems* – early attempts attempted to identify objects based upon colour alone. This produced too many false positives – all behaviours were relevant at all times due to the high number of imaginary objects.

- *Shape only recognition* – objects were marked with black shapes on white. This worked well on a stationary observer, but failed once the robot started to move. The motion of the observer blurring the shape into a grey blob.

  Additionally, shapes with sharp corners could not be cleanly recognised, as the contour would not always travel around the corner. This severely limited the range of shapes.

- *Range from Magnification* – an early version of range detection was to compute range based upon the ratio of projected size of the object and the known actual size. Measurements were taken of the standard object marker at a number of known ranges and an interpolated look-up table generated. In the event, results from this method were very different between cameras, it was easier to calibrate the cameras, recover the exact focal-lengths and use the method described in Figures B.8 and B.9.

B.1.5 Motors and Actions

Locomotion

Hook

The original robots were equipped with only a wide scoop as a gripper. The idea was that the complexity and cost of a powered gripper could be avoided, the robots' anticipated environment was a flat two dimensional plane so they could

manipulate objects by simply pushing them wherever they need to go. As there was no third dimension, there would be no need for powered manipulators.

Unfortunately, this was too optimistic. Although the scoop scheme worked well in prototype form, as soon as the robots' environment was bounded by the arena walls[3] the robots lost all control of their captured objects upon encountering the wall. This meant that all target objects eventually washed up against the arena walls, from where they could not be retrieved, unless they were continually re-set.

Clearly, this was inadequate. At best it was inconvenient and at worst it actively damaged the swarm's chances of completing its task. To solve this problem a micro-servo was installed behind the scoop (see Figure B.10 and Appendix B). A thin wire (a bent paper-clip, actually) was attached to the horn to act as a "hook" with which a captured object could be snagged. Power was provided by the master computer battery and control from the main computer's sole hardware-PWM capable pin. Control was activated via the grip and un-grip actions, these were the only actions allowed to access the hook control. Only two positions, HOOK_UP and HOOK_DOWN were supported.

This primitive set-up allowed for the robots to hold on to captured objects during sharp turning, reversing and avoidance.

Actions

The low speed required to overcome the motion blur problem necessitated fairly primitive control algorithms. Eventually (after alternatives implementations failed), all control for the "move to" and "move from" actions were based on target object bearing exclusively. The object bearing values were divided into six zones: less than one degree, one to six degrees and greater than six degrees (with mirroring negative values). Appropriate motor values were selected for each zone based upon the maximum motor speed multiplied by a constant. The maximum motor speeds and constants were determined via trial and error.

The grasp action was also implemented simply. The robots moved forward until a captured-object appeared in the object list. The hook (if implemented – see below) was then lowered and the action returned success. The action timed out if the "move forward" component went on for longer than ten seconds. A half second movement was guaranteed. These times were based upon experience with the robots and how far (roughly) they could move in a given time and for a given power setting. Better implementations would have closed-loop feedback and move specific distances (see Section 5.1.3) wheel encoders were not available for the real-world robots and not used in the simulator.

Dropping objects was accomplished in a similar fashion. The hook (if present) was raised, the robot backed up and then executed an on-the-spot turn to remove the dropped object from sight. Backing and turning was also based upon time.

To execute avoidance the real-world robots began by backing up slightly, to clear their touch sensors from the obstacle (if the obstacle is not behind). The control loop then found the highest confidence object within the avoidance distance and performed one of four avoidance options (move forward, back, left or right) based on the quadrant in which that object existed. This was done

[3]See Section 4.2.4 for a discussion on why this was necessary

continuously until no objects existed within the avoidance distance.

These are crude, ad-hoc methods, but were ultimately faster and easier to implement than any other. The results were surprisingly good, although the real-world robots' response was fairly sluggish and turning circle somewhat large.

B.1.6 Clocks and Timing

Real-world timekeeping was inaccurate. In the simulator it was possible to fix the time exactly, in terms of simulation steps. In the real-world this was much more approximate. Although each robot was equipped with an on-board clock, this was simply the computer's system clock and not calibrated to an external source or to other robots in the swarm. As a result all robots had different, unknown starting times. Simultaneous initialisation of the experiment was achieved by simultaneously removing an obstacle (a paper cover) from the robots' IR gripper sensor with a string. Theoretically, it should have been possible to then clock any desired experimental time from this common point – see Section C.1.1 for more on this.

Unfortunately, in practice, it was found that the robots' internal clocks were extremely inaccurate over a long period. It is believed that this is a function of the high computational load placed upon the CPU by the requirements of real-time vision processing. The robots' system clocks ran faster than the wall-clock, with a shorter "second". This could be by as much as 10%, and never exactly the same in any two robots.

The result was a staggered experiment termination, with robots stopping singly over an approximately two minute interval.

Clearly this was insufficient, but it could not be fully overcome. External timekeeping was not practical: the robots had neither external clocks or wireless connections and there were no resources with which to procure or time to install such devices.

Instead, we used a stopwatch to time the actual duration of experiments, from initialisation to when the first robot stopped or the target object was removed. As we shall shortly see, in experiments where duration time was an important factor, we will use both the internal clock and the stopwatch "wall-clock" times.

For times that were recorded as the summation of very small increments over a non-continuous period time (as were behaviour threshold times, object hold times, etc.), this effect was believed to be minimised. If the clock error was a constant percentage of each second, division by the experiment duration as recorded by the robot should have cancelled the error.

Nevertheless, we must stress that because of this, pinpoint temporal accuracy in real-world robots was impossible. We shall use time extensively in computing results, but these are not intended to be universally precise, but simply a guide to observations.

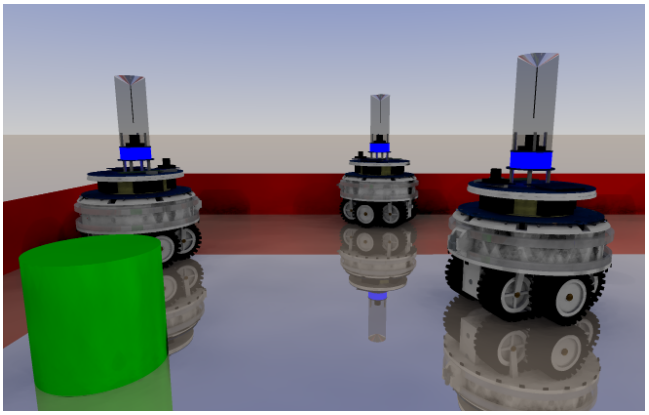

Figure B.11. Three simulated robots and the target-obj-1 ("ball") object. The cylindrical protuberance on the top of the robots is an omni-directional camera, with the blue robot beacon at the base. The tiny "gripper" is just visible on the right-hand robot as a slight bulge on the middle section facing the target object.

B.1.7 Chassis

B.2 Simulation

B.2.1 Overview

The simulated robots are treated as ideal approximations of the "perceive, manipulate, locomote" swarm agent model. The perception system is a perfect camera, that returns the type and location of objects relative to the agent. Locomotion is in two dimensions via a differential drive and manipulation governed by a ideal pincer gripper.

It is important to note that the simulated Architecture was realised via a different source code than that used in the real-world machines. The real-world machines make extensive use of shared memory, the OS scheduler and the `fork()` system call to achieve parallelism in the various Architecture subprocesses (see Chapter 3). The simulated robots rely on a superabundance of computing power and the luxury of non real-time implementation. The simulation step process made it easier to simply re-write The Architecture as a sequential system with each major Architecture process as a method of a larger swarm agent class. These methods were executed in a specific order (sensors, behaviour selection, behaviour execution, survival selection and survival execution) once per simulation step.

In theory, it would have been possible to make The Architecture code fully portable and, had this been done at the start, it might have been easier. However, by the time it was fully implemented, it had been created in two distinct streams and reconciliation would have taken far too much time. A side effect of this is slight differences in minor Architecture features, the effect of which we shall discuss presently.

B.2.2 Perception

The simulated robots possessed a coloured "beacon", which were illuminated in blue for agent object identification. Target objects are identified in the simulation with various coloured "LED's" with object recognition being an idealised version of blob detection. The "camera" returned a list of object blobs, exploiting the geometrical properties of the simulated omni-directional camera to compute their exact location relative to the robot. To mirror the real-world robots the field of view was restricted. Objects that fell outside the cone described by an angle of $+/-30$ degrees and a range of one metre from the edge of the robot were ignored. This represents a slight divergence from the real-world robots (which, in practice have a narrower field of view).

A ring of proximity sensors was also present. This enabled the robots to reliably detect objects up to about 17cm of the robot. Results from the forward proximity sensors were ignored if an object was present in the gripper – although it is suspected that this filtering was not always successful as robots could be occasionally observed turning in circles as though permanently avoiding collision. This switch off also implies that the simulated robots are blind to proximity objects approaching from the front quarter when carrying a captured object.

There was no noise in any of the simulated sensors.

B.2.3 Actions

In simulation, the robots did not possess a very sophisticated control loop. Instead they simply had their motors fixed at an appropriate speed and were permitted to "waddle" towards target objects; if the target was on the left, the left-hand motor was stopped and the right-hand motor left running. If left to their own devices they would (eventually) converge on an object. Although this had a tendency to make the robots look like a line of fat ducks, for the sake of simplicity such a sacrifice of dignity and precision was considered acceptable.

Simulated robots did posses a powered gripper (see Figure B.11), but in our implementation this was simply treated as a magical appendage that allowed captured objects to "stick" to the robot and un-stick on command. Through trial and error, a distance was found where the gripper was effective. The grasp action moved the robot to this distance from the target object and called the appropriate method to close the gripper's jaws. Dropping was the same in reverse, followed by a sharp turn to remove the dropped object from the robot's field of view.

We shall discuss the simulated robots in greater detail in Section C.1.2. At this stage, they can be considered an approximation of the real-world robots and are the same conceptually unless otherwise mentioned.

Appendix C

Swarm Implementation

C.1 Real-World and Simulation

The Architecture was implemented on two distinct swarms: one comprised of the real-world robots (above) and a virtual swarm generated by ARGoS simulator developed by Pinciroli, Trianni, O'Grady, Pini, Brutschy, Brambilla, Mathews, Ferrante, Di Caro, Ducatelle, Birattari, Gambardella & Dorigo (2012).

It should be stressed that the simulation was not intended as a direct reproduction of the real-world and that the simulated robots represent quite different machines to those eventually realised in hardware. In fact, they are only similar in that they are both swarm agents in the move, manipulate and perceive sense discussed previously. As The Architecture was intended to be general, it was imperative that it be operable on multiple platforms.

While the same tests were undertaken in both environments, they had distinct purposes. The simulation was used for:

- Development, verification, debugging and testing of early Architecture concepts.
- Recording of reliable numerical data on micro agent and macro swarm performance.
- Replicating the basic tasks on a very large-scale swarm, to illustrate The Architecture functioning on a scale fiscally impossible in the real-world.

While the real-world robots were used to:

- Test The Architecture in a less-idealised environment.
- Illustrate the functionality of The Architecture on a similar, yet physically different swarm.
- Validate the simulation by achieving comparable results under these conditions.

The differences between the real-world and simulator implementations of The Architecture provided a valuable insight into the practicalities of implementing

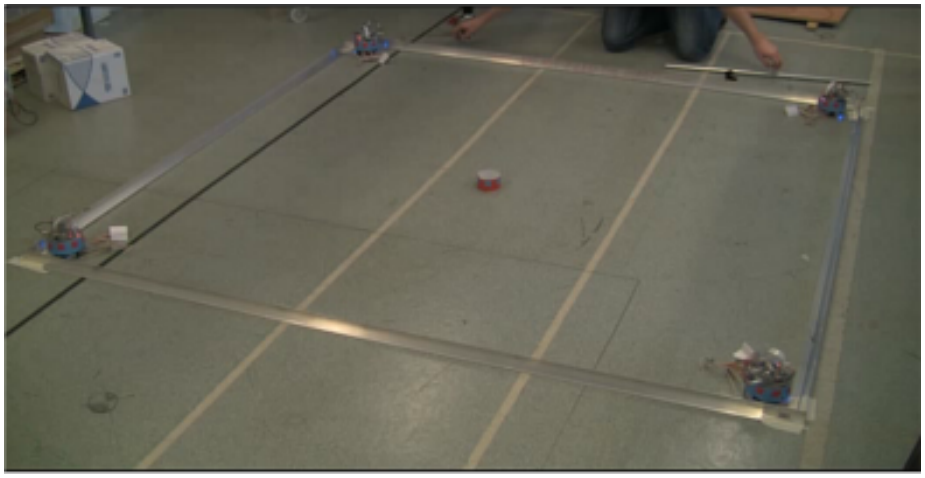

Figure C.1. The initial set-up of the ball passing task, as the infrared initialisation barriers are removed. The lines on the floor are not related to this project. From ball passing experiment 1.

a universal swarm architecture and the sensitivity of swarm performance to the unavoidable differences between swarm hardware and environments, as we shall see.

However, despite the inherent differences between the two environments, the tasks executed by both swarms were identical and wherever possible, the two environments were made similar in gross features.

Before progressing to the actual results, we shall conduct a brief examination of the two worlds.

C.1.1 The Real-World

The Robots

As we have discussed previously, the physical robots went through several distinct phases. All tests were conducted on machines that had infrared captured object detection and proximity touch sensors. However, initial tests of the ball-passing (experiments 1 through 12 inclusive) were conducted with "scoop" robots featuring the more primitive touch sensor incarnation. Hooks and wire "antennae" type sensors are present for experiments 13 through 20 inclusive. Primitive grouping1 experiments were also conducted with the scoop robots, although these were not recorded and are considered debugging trials, only briefly commented on. Grouping2 experiments were always conducted with the last hardware incarnation.

The Arena

The real-world robots were arranged in a square arena 2.4 metres to a side[1] (see Figure C.1) with one robot in each corner, all facing in towards the target

[1]The real-world dimensions of the arena were chosen unscientifically, 2.4 metres gave a "good" amount of space proportional to the robots' physical size, fitted well in the available

object at the centre. In the real-world arena, these positions were approximate.

The arena was bounded by metal barriers fixed to the floor and could not be shifted by the robots (liberal application of tape ensured this). The barriers were physical obstacles that were invisible to the robot's vision sensors and did not trigger the infrared gripping sensor.

The arena floor was a flat, hard vinyl surface, presumably laid over concrete. Imperfections in the form of fine dirt, small bumps and marked lines – relics of older projects which can be seen in Figure C.1 – were present. As far as could be determined, these had little effect on experimental results and were largely ignored, although a build-up of rather noxious fluff in the robots castors and around the motor axles was noticeable over time. This was periodically cleaned with a sharp point and lead to a slight, but largely undetectable and probably insignificant, increase in motor traction.

All robots were started simultaneously, using the gripper sensor as a signal; a obstacle (a paper barrier) was placed across the sensor and the architecture initialisation sequence was set to hang until the sensor cleared (the sensor needed to be blocked before The Architecture booted). All of these paper obstacles were removed simultaneously with a string, providing an approximately simultaneous initialisation. This process is being executed in Figure C.1. This process was not exactly precise, but deemed sufficient for real-world experimentation.

In all real-world experiments the arena was unchanged. The only modification was the introduction of multiple target objects in the grouping1 task and the addition of the central pillar object in the grouping2 task. In the former case, eight target objects were arranged around the edges of the arena, in a roughly even distribution. For the grouping2 task this arrangement was duplicated with the addition of four target-obj-2 type objects in the centre to act as the central pillar. Four such objects were found to be necessary to provide good visual identification at all orientations and prevent occlusion.

Procedure

Real-world experiments were all started by the approximately simultaneous removal of a paper barrier from the robots' infrared sensors (see Figure C.1 with a string. They were concluded by manually powering off the robots and (in the ball passing task) the removal of the target object.

The real-world robots did not exhibit the robustness of their virtual counterparts. Mechanical failures, stalls and entanglements were common. The robots' physical construction was simple and their touch-sensors unreliable (as contact switches, they only functioned if pressed from the correct direction). In most cases, the snarls that resulted from several robots colliding in close proximity was too much for the primitive avoidance function to resolve.

In these cases, rather than permit the motors to strain at stall indefinitely, the experimenter was compelled to untangle the robots manually. Wherever possible this was done at a distance with a pole (although some collisions unseated the printed chassis and needed to be re-set by hand – as can be seen in the attached recordings). This manual re-set was also employed on the not infrequent occasions when the touch sensors failed to trigger upon encountering

laboratory space and was readily available off-the-shelf from the local hardware. The barriers are actually an aluminium 'T' section approximately 40×40 mm along the principle dimensions.

a wall or other obstacle. Improvements in touch-sensor design reduced the need for this intervention, but ultimately the fundamentally jury-rigged nature of the proximity sensors could never be fully overcome.

The other necessary intervention was to keep the target objects clear of the arena barrier wall. Even hook equipped gripping robots did not have the ability to retrieve the ball if it was hard up against the wall – their forward avoidance sensors would trigger if they approached the target object head-on. Scoop robots, of course, could not back-up with a captured object, rendering such recovery impossible. Setting proximity objects as an avoidance exception during the acquire behaviour was not a solution, as this tended to cause the robots to stall against the wall if the grip was not perfect. Although, one must admit that this avenue was not fully explored.

Instead, the target object was pushed away from the wall manually. The earliest experiments with the scoop robots required that it be re-set in the centre of the arena every time it touched the wall. This was a very unsatisfactory state of affairs that inspired the development of powered hooks. Later, hook equipped robots were more autonomous and the ball was simply pushed off the wall with a screwdriver (which provided a constant short distance), or, later still, shoved a few centimetres with a pole.

These interventions are clearly less than optimal as they introduce a potential for bias into the experiment. However, no realistic alternatives were available. It is the author's belief that, in the hook equipped passing task especially, they do not occur with sufficient frequency to be a defining factor in the result.

However, they do appreciably add to the confusion in the grouping1 task, as we shall see.

We should also note that the quantitative results for the real-world ball passing task represent an approximation at best. Because of the clock errors above and the method used to gauge the presence of a captured object is vulnerable to false positives, the recorded results will lean towards over-estimation.

This problem was not considered significant as the observation of the robots' behaviour tallied approximately with the recorded data. However, it should be noted that the real-world hold time data is intended to be only used as an approximate guide.

C.1.2 Simulation

The Arena

The simulation arena was identical in shape, but scaled in area to be proportional to the size of the robots. Real-world robots fitted into a bounding box of approximately 23 by 16 centimetres (in their penultimate stick-touch sensor configuration) while the simulated robots were more circular in configuration with a diameter of 17cm. This gives a total area of approximately 368 and 227 square centimetres, respectively. To keep the simulated arena proportional to the robots' footprint the simulated walls were 0.933 metres to a square side.

The simulated target objects were set to 50 millimetres in diameter (height seemed irrelevant, the simulated "LED" markers did not seem to suffer from significant occlusion and the omni-directional camera projected significantly from the robots like a ship's funnel). The actual dimensions of the simulated objects were considered largely irrelevant, as they appeared to be able to be bunched

tightly together in defiance of their physical dimensions. This is believed to be a bug in the simulator. All simulated target-obj-1 objects were set to one gram in mass (mimicking the essentially mass-less paper objects in the real-world) and set to `movable=''true''`. Target-obj-2 objects in the grouping2 task are thinner, higher and absolutely immovable.

For tasks requiring more than one target object, the initial distribution of target objects was "uniform random" throughout the arena. The robots were always instantiated in the arena corners orientated to the centre of the arena (see Figure 4.3). In small-scale simulations ten target objects were used. This is in contrast to the eight used in the real-world. This discrepancy is due to the cluttered appearance of the real-world arena when more than eight objects were used. Objects could have been scaled, shrunk (probably impractical in the real-world due to the practicalities of object recognition) or enlarged in simulation, but ultimately, this was considered unimportant as we do not compare the task execution speed or efficiency in detail, we merely wish to see that The Architecture functions. Thus a subjectively "good" number of target objects was used.

In addition a much larger-scale simulation was attempted. In the large-scale version, the simulation arena was expanded to twenty by twenty metres and seeded with 800 target objects and 100 robots. Both were distributed in a uniform random distribution, with the robots' orientation also randomised (see Figure 4.6, left).

Procedure

The simulated robots all started simultaneously when the simulation run began. They run without interference until it stopped. Simulation length was controlled exactly. Simulation length was varied as convenient, but all simulations ran at ten simulation steps per second.