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:

$$P_{i,j} = \frac{s_{i,j}^n}{s_{i,j}^n + \theta_{i,j}^n} \quad (1)$$

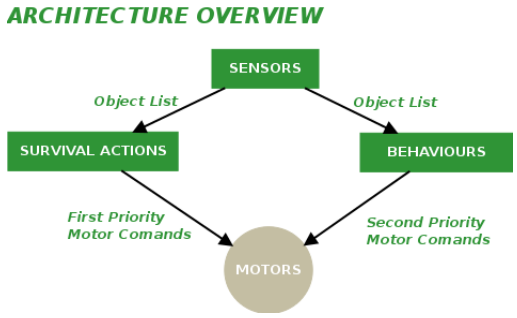

Figure 1: An overview of the architecture showing the three parallel spheres of operation and the Object List linking them.

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.

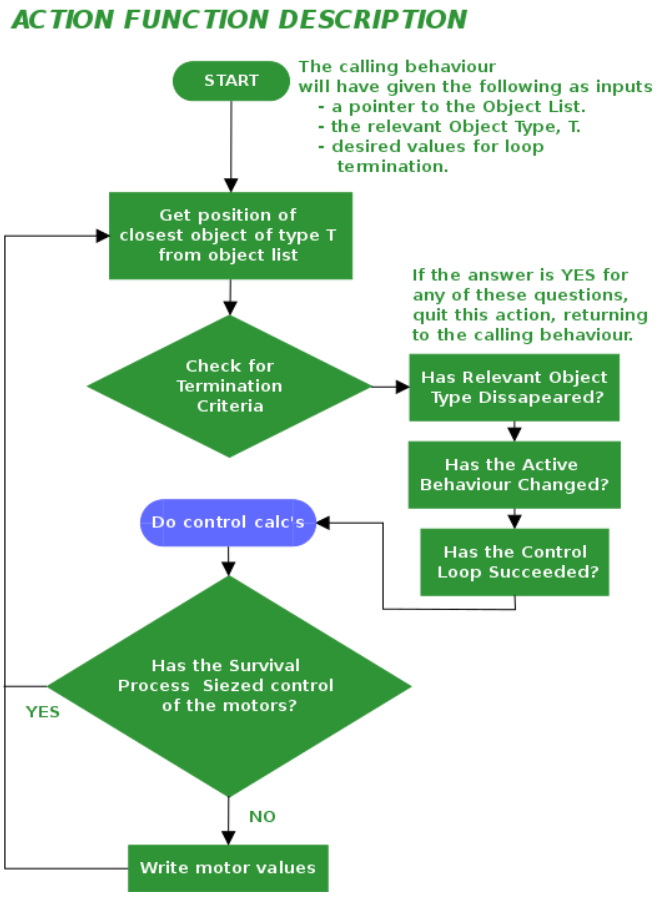

Figure 2: All motor action functions follow this basic template. The Green Blocks represent sections that are universal, only the rounded boxes are specific to each action. The purpose of action functions is to determine the current motor values for the effectors.

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

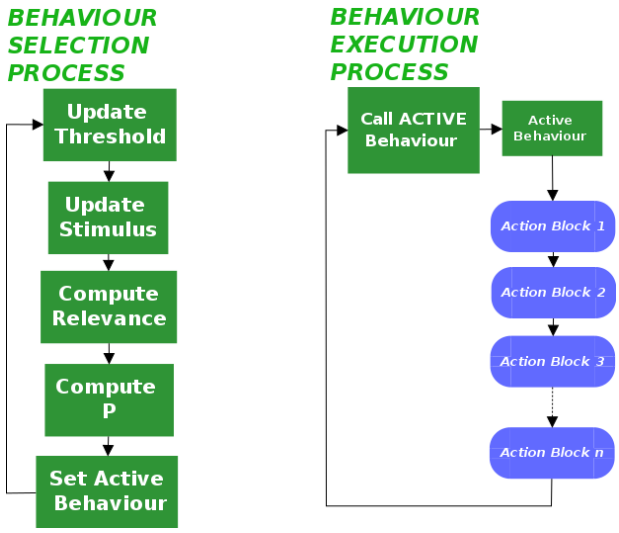

Figure 3: The behaviour selection and execution processes. Note that these operate in parallel, behaviour selection is continuously running. Actions are obliged to recognise when the active behaviour has changed and terminate. All elements of this diagram have access to the Object List. The rounded boxes 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

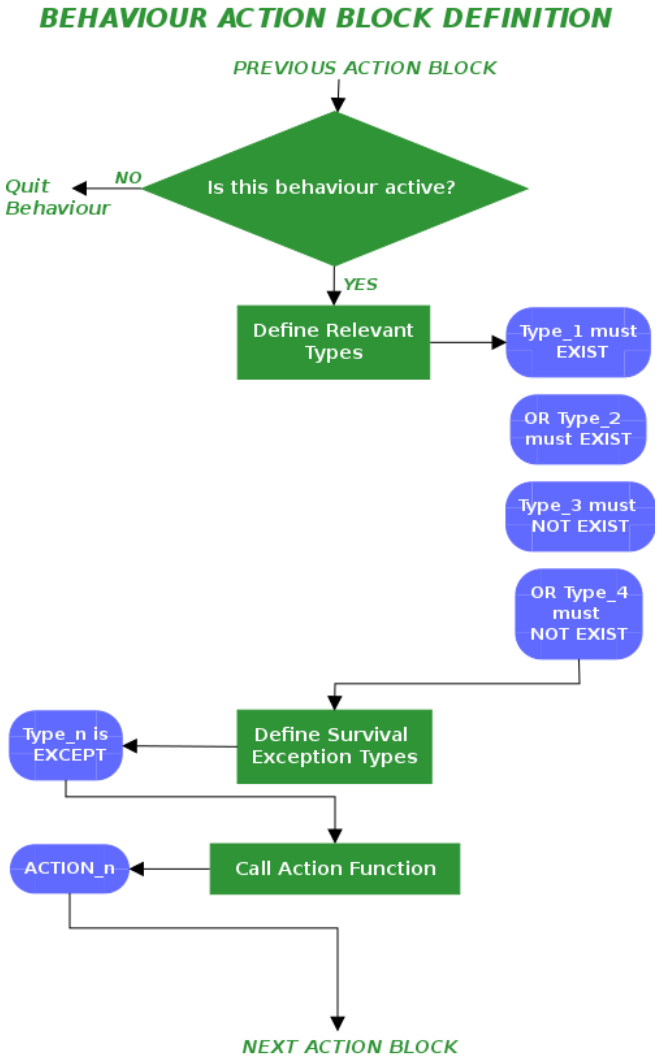

Figure 4: The action block format. The square boxes illustrate what is required, the rounded boxes what the user may modify for each action block. Note that it is not necessary to use all possible combinations of *exist* and *not_exist*. Any number of object types with any number of logical conditions 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

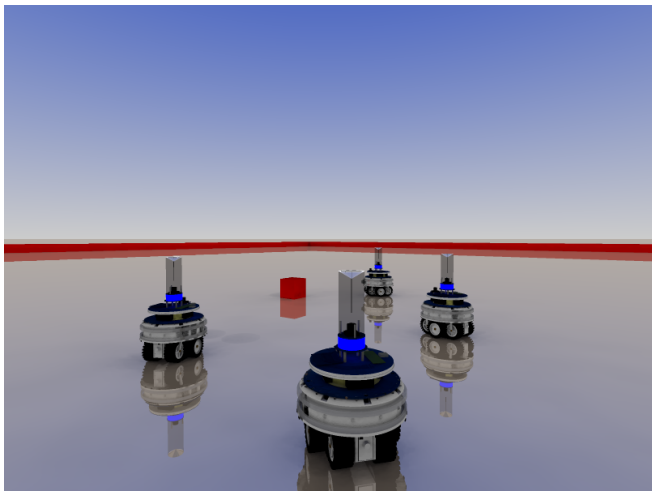

Figure 5: Still from a simulation run showing four robots and the 'ball'

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

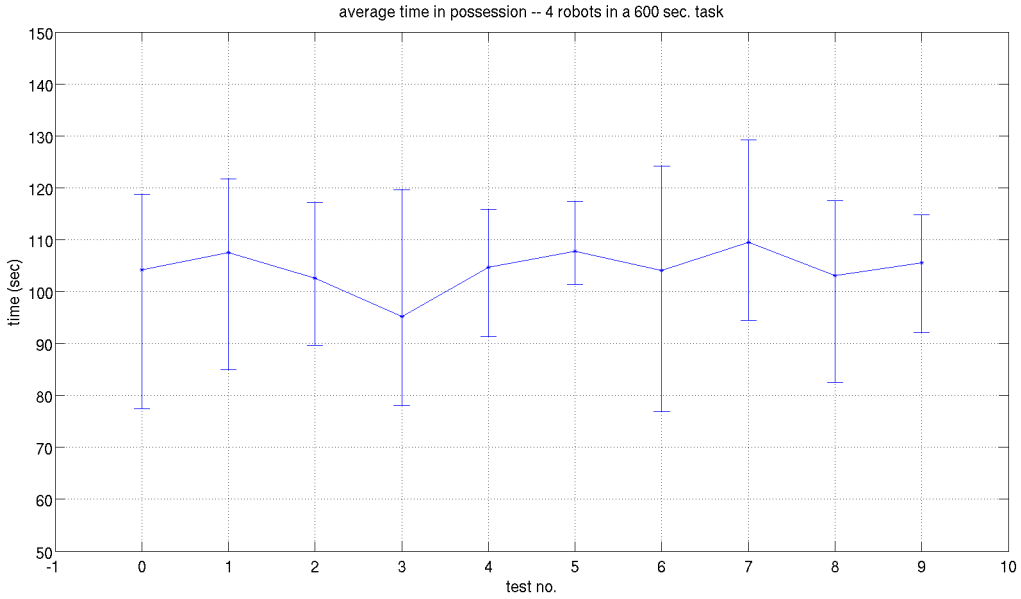

Figure 6: The results of 10 simulations for a four robot group. This plot shows the average time each robot in the group has the ball (seconds) in each 600 second simulation. The maximum and minimum times for each simulation are also shown.

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.