Self-regulated Multi-robot Task Allocation

Md Omar Faruque Sarker

University of Wales, Newport

A dissertation submitted to the University of Wales, Newport
for the degree of Doctor of Philosophy

Robotic Intelligence Lab,
Department of Computing,
Newport Business School,
Allt-yr-yn Campus, Allt-yr-yn Avenue,
Newport, NP20 5DA
United Kingdom.

Email: Mdomarfaruque.Sarker@students.newport.ac.uk

July 4, 2010

Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

1.1 Multi-robot task allocation (MRTA)

Robotic researchers generally agree that multiple robots can perform complex and distributed tasks more conveniently. A multi-robot system (MRS) can provide improved performance, fault-tolerance and robustness in those tasks through parallelism and redundancy (**?**, **?**, **?**). In last two decades, many robotic researchers have conducted many challenging researches on various aspects of MRS. Results of those researches have been applied in many real robotic applications. For example, MRS can be used in: large space monitoring tasks, e.g underwater environment monitoring, dangerous tasks e.g., robotic de-mining, tasks that scale-up or scale-down over time, e.g. factory automation, and so forth.

In order to get potential benefits of MRS in any application domain, we need to solve a common research challenge. *How can we allocate tasks among multiple robots dynamically ?* In robotic literature, this issue is identified as the *multi-robot task allocation* (MRTA). This problem can also be treated as the *division of labour* (DOL) among robots, analogous to the DOL in biological and human social systems. Although the term “division of labour” is often used in biological literature and the term “task-allocation” is primarily used in multi-agent literature, both of these terms carry similar kinds of meanings with slightly different semantics. In this dissertation, we have used these terms interchangeably.

MRTA is generally identified as the question of assigning tasks in an appropriate time to the appropriate robots considering the changes of the environment and/or the performance of other team members (**?**). This is a *NP-hard* optimal assignment problem where optimum solutions can not be found quickly for large and complex problems (**?**).

The complexities of the distributed MRTA problem arise from the fact that there is no central planner or coordinator for task assignments, and in a large MRS, generally robots have limited capabilities to sense, to communicate and to interact locally. None of them has the complete knowledge of the past, present or future actions of other robots. Moreover, they do not have the complete view of the world state. The computational and communication bandwidth requirements also restrict the solution quality of the problem (**?**).

Researchers from multi-robot and multi-agent systems, operations research and other disciplines have approached the issue task-allocation in many different ways. Traditionally, task allocation in a multi-agent systems has been divided into two major categories: 1) Predefined and 2) Bio-inspired self-organized task-allocation (**?**).

Early research on predefined task-allocation was dominated by intentional coordination (**?**), use of dynamic role assignment (**?**) and market-based bidding approach (**?**). Under these approaches, robots use direct task-allocation method, often to communicate with group members for negotiating on tasks. These approaches are intuitive, comparatively straightforward to design and implement and can be analysed formally. However, these approaches typically works well only when the number of robots are small (≤ 10) (**?**).

On the other hand, self-organized task-allocation approach relies on the emergent group behaviours, such as emergent cooperation (e.g. **?**), adaptation rules (e.g. **?**) etc. They are more robust and scalable to large team sizes. However, most of the robotic researchers found that self-organized task-allocation approach is difficult to design, to analyse (formally) and to implement in real robots. The solutions from these systems are also sub-optimal. It is also difficult to predict exact behaviours of robots and overall system performance.

Within the context of the Engineering and Physical Sciences Research Council (EPSRC) project, “Defying the Rules: How Self-regulatory Systems Work”, we have proposed to solve the above mentioned self-regulated DOL problem in an alternate way (**?**). Our approach is inspired from the studies of emergence of task-allocation in both biological and human social systems. These studies show that a large number of species grow, evolve and generally continue functioning well by the virtue of their individual self-regulatory DOL systems.

The amazing abilities of biological organisms to change, to respond to unpredictable environments, and to adapt over time lead them to sustain life through biological functions such as self-recognition, self-recovery, self-growth etc. It is interesting to note that in animal societies task-allocation has been accomplished years after years without a central authority or an explicit planning and coordinating element. Direct and indirect communication strategies are used to exchange information among individuals (**?**).

The decentralized self-growth of Internet and its bottom-up interactions of millions of users around the globe present us similar evidences of task-allocation in human social systems (**?**). These interactions of individuals happen in the absence of, or in parallel with, strict hierarchy. Moreover from the study of sociology (e.g. **?**), cybernetics (e.g. **?**), strategic management (e.g. **?**) and related other disciplines we have found that decentralized self-regulated systems exist in nature and in man-made systems which can grow and achieve self-regulated DOL over time by the virtue of their common bottom-up rules of self-regulation.

From the above mentioned multi-disciplinary studies of various complex systems, we have proposed four generic rules to explain self-regulation in those social systems. These four rules are: *continuous flow of information*, *concurrency*, *learning* and *forgetting*, all of them will be explained in Chapter 4. Primarily these rules deal with the issue of deriving local control laws for regulating an individual's task-allocation behaviour that can facilitate the DOL in the entire group. In order employ these rules in the individual level, we have developed a formal model of self-regulated DOL, called the *attractive field model* (AFM).

AFM provides an abstract framework for self-regulatory DOL in social systems. However, without a proper validation in a real robotic system, we can not claim its applicability to the above MRTA problem. Particularly, we have got the following two questions:

1. Can AFM be adopted to solve the above MRTA problem for a relatively large

group of real robots ?

2. Can this abstract model be used in complex multi-tasking environments, preferably under an industrial scenario, rather than emulating the canonical biological tasks, e.g. foraging ?

The first question demands to prove the very basic capabilities of AFM, i.e. whether it can provide the basic form of DOL under a large group size. As we know that conducting real robotic experiments with large number of robots is time consuming and resource intensive, in this study we use the term “large group of robots” to mean a group with at least 10 or more robots [1]. Thus we have hypothesized that AFM can be used as a distributed task-allocation mechanism for a large group of robots. It should be capable of providing all the basic task-allocation facilities, e.g. plasticity in task-allocation, task-specialization, as discussed in detail in Sec. 2.1.3.

The second question put AFM under a big challenge. Traditionally, robotic researchers, who use self-organized task-allocation methods, limit their experiments to test if their models can sufficiently imitate biological tasks, e.g. foraging (e.g. **?**), pre-retrieval (e.g. **?**). These experiments simply match the biological DOL models with their robotic counterparts. However, we have hypothesized that AFM can work under real task settings where multiple tasks should be served by multiple-robots concurrently. We believe that even if we do not perform specific complex industrial tasks by our robots, we can use AFM to find the high-level task-allocation performance by mimicking any suitable industrial task. This high-level task-allocation performance can be used as an indication of the validity of AFM for putting it in real-world applications.

The outcome of our research can be applied to solve generic task-allocation problem in numerous multi-agent systems. Generally AFM can be used as a generic framework for DOL in various social systems and particularly, our implementation can be used as a guide to design MRTA solution of large MRS in order to perform real industrial tasks. As an example, our technique can be useful in automated material handling tasks in warehouses or in manufacturing industries which face

[1]Although this is very difficult to specify a certain number for defining “large”, we find a consensus with some other researchers, e.g. (e.g. **?**) that a large group of robots currently stands for roughly 10 or more robots.

all the existing challenges of traditional centralized and sequential manufacturing processes e.g. changing the manufacturing plant layouts on-the-fly, adapting for high variation in product styles/quantities, and dynamic allocation of active manufacturing resources e.g., robots and automated guided vehicles (**?**).

1.2 Communication & sensing strategies for MRTA

In biological social systems, communications among the group members, as well as sensing the task-in-progress, are two key components of self-organized DOL. In robotics, existing self-organized task-allocation methods heavily rely upon local sensing and local communication of individuals for achieving self-organized task-allocation. However, AFM differs significantly in this point by avoiding the strong dependence on the local communications and interactions found in many existing approaches to MRTA. AFM only requires a system-wide continuous flow of information about tasks, agent states etc. But this can be achieved by using both centralized and decentralized communication modes under explicit and implicit communication strategies. Chapter 2 reviews these communication modes and strategies with necessary details.

In order to enable continuous flow of information in our MRS, we have implemented two types sensing and communication strategies inspired by the self-regulated DOL found in two types of social wasps: 1) *Polistes* and 2) *Polybia* (**?**). Depending on the group size, these species follow different strategies for communication and sensing about tasks. Polistes wasps are called the *independent founders* (IF) in which reproductive females establish colonies alone or in small groups (in the order of 10^2), but independent of any sterile workers. On the other hand, polybia wasps are called the *swarm founders* (SF) where a swarm of workers and queens initiate colonies consisting of several hundreds to millions of individuals.

The most notable difference in the organization of work of these two social wasps is: IF does not rely on any cooperative task performance while SF interact with each-other locally to accomplish their tasks. The work mode of IF can be considered as *global sensing - no communication (GSNC)* where the individuals sense the task requirements throughout a small colony and do these tasks without communicating with each other. On the other hand, the work mode of SF can be treated

as *local sensing - local communication (LSLC)* where the individuals can only sense tasks locally due to large colony-size and they can communicate locally to exchange information, e.g. task-requirements (although their exact mechanism is unknown).
In this study, we have used these two sensing and communication strategies to compare the performance of the self-regulated DOL of our robots under AFM. Our research questions behind putting these strategies into action are as follows.

1. Is this the case that task-allocation through GSNC strategy should be limited only to small group of robots, like polistes wasps ?

2. Does the task-allocation performance in large groups significantly vary under GSNC and LSLC strategies ?

Regarding the first question, if the self-regulated DOL in a large group of robots can not be achieved by following GSNC strategy, we can conclude that this strategy is inappropriate for a large group. The second question quantitatively investigates the variations of performance due to the use of different communication and sensing strategies. The findings from this comparative study of different bio-inspired communication and sensing strategies can be very useful for designing efficient MRTA solutions. Thus we can be able to conclude, both quantitatively and qualitatively, whether local communication and local interaction should be considered as the prerequisites to achieve self-regulated DOL in large group of robots.

1.3 Contributions

The main contributions of this thesis are as follows:

- Introduction of attractive field model (AFM), an inter-disciplinary generic model of division of labour, as a basic mechanism of self-regulated MRTA.

- Validation of the model through experiments with reasonably large number of real robots (i.e. 16 e-puck robots).

- Comparisons of the performances of two bio-inspired sensing and communication strategies in achieving self-regulated MRTA.

- Development of a flexible multi-robot control architecture using D-Bus inter-process communication technology.

- Classification of MRTA solutions based on three major axes: organization of task-allocation, interaction and communication. This concise classification helps us to remove the ambiguities among various MRTA solutions found in the literature.

1.4 Thesis outline and relevant publications

This dissertation has been organized as follows. Chapter 2 reviews the related literature on general terms, key issues of MRS and MRTA. This chapter includes a concise review of biological communication and sensing strategies for self-regulated task-allocation. This chapter also focuses on discussing the related work on communication in MRS within the context of our MRTA problem.
Chapter 3 describes our research methodology with a review of experimental tools and technologies used in this dissertation.
Chapter 4 describes the validation of attractive filed model in our MRS under the GSNC strategy. It elaborately discusses the findings from our self-regulated MRTA experiments.
Chapter 5 presents our extended work on using AFM under the LSLC strategy and analyses the performance of self-regulated task-allocation under both GSNC and LSLC strategies.
Chapter 6 concludes this dissertation with a summary and future research directions.
The following two publications are relevant to this dissertation.

- Sarker, M. & Dahl, T. *A Robotic Validation of the Attractive Field Model: An Inter-Disciplinary Model of Self-Regulatory Social Systems*. In Proc. of the Seventh International Conference on Swarm Intelligence (ANTS 2010) to appear, 2010.

- Sarker, M. & Dahl, T. *Flexible Communication in Multi-robotic Control System Using HEAD: Hybrid Event-driven Architecture on D-Bus*. In Proc. of the UKACC International Conference on Control 2010 (CONTROL 2010), to appear, 2010.

CHAPTER 2

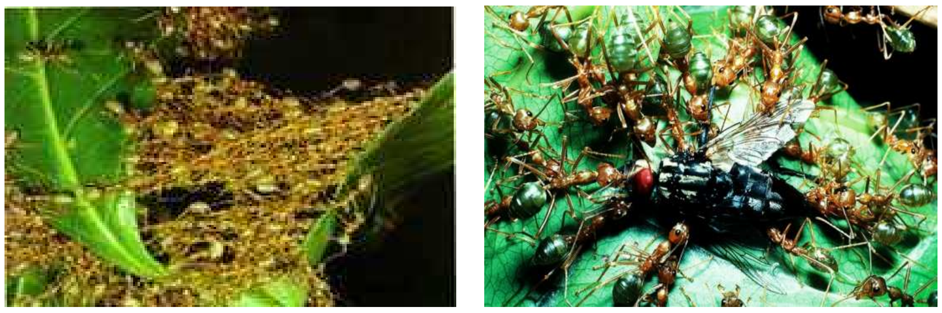
Background and Related Work

2.1 Definition of key terms

2.1.1 Self-regulation

Animals and flying beings, that live on or above earth, form social communities similar to human societies (**?**). In recent years, the biological study of social insects and other animals reveals us that simple individuals of these self-organized societies can solve various complex and large everyday-problems with a few behavioural rules, relying on their minimum sensing and communication abilities (**?**). Some common tasks of these biological societies include: dynamic foraging, building amazing nest structures, maintaining division of labour among workers (**?**). These tasks are done by colonies ranging from a few animals to thousands or millions of individuals. Despite their huge colony size, they easily achieve surprising efficiency in those tasks with many common features, e.g. robustness, flexibility, synergy (for an example in ants, see Fig. 2.1). Today, these findings have inspired scientists and engineers to use this knowledge of biological self-organization in developing solutions for various problems of artificial systems, such as routing traffics in telecommunication and vehicle networks, designing control algorithms for large groups of autonomous robots, automating industrial shop-floor tasks and so forth (**?**).

Self-organization (SO) in biological and other systems are often characterized



(a) Weaver ants (b) Green-tree ants

Figure 2.1: (a) During nest construction, weaver ants combine two leaves by pulling them from two sides, reproduced from **?**. (b) Green-tree ants retrieve a relatively large prey. From http://www.spacecollective.org, last seen on 01/05/2010.

in terms of four major ingredients: 1) positive feedback, 2) negative feedback, 3) presence of multiple interactions among individuals and their environment, and 4) amplification of fluctuations e.g., random walks, errors, random task-switching (**?**). An external observer, as if looking through transparent glasses, can recognize a self-organized system by observing the individual interactions of that system from four interlinked perspectives (Fig. 2.2). The first perspective is the *positive feedback* or amplification that can be resulted from the execution of simple behavioural "rules of thumb". For example, recruitment to a food source through trail laying and trail following in some ants is due to positive feedback that attract other ants to follow the trail and to lay more pheromones over time. The second perspective is the *negative feedback* that counterbalances positive feedback. This usually occurs to stabilize collective patterns, e.g., crowding at the food sources (saturation), competition between paths to food sources etc. The third perspective is the *presence of multiple interactions* that can be direct peer-to-peer (P2P) or indirect *stigmergic*, e.g. ants pheromone laying. Finally, the fourth perspective is the *amplification of fluctuations* that comes from various stochastic events. For example, errors in trail following of some ants may lead some foragers to get lost and later on to find new, unexploited food sources and recruit other ants to those sources.

In a self-organized system, an individual agent may have limited cognitive, sensing and communication capabilities. But they are collectively capable of solving complex and large problems, e.g. coordinated nest construction of honey-bees (Fig. 2.4), collective defence of school fishes upon predator attack (Fig. 2.5), or-

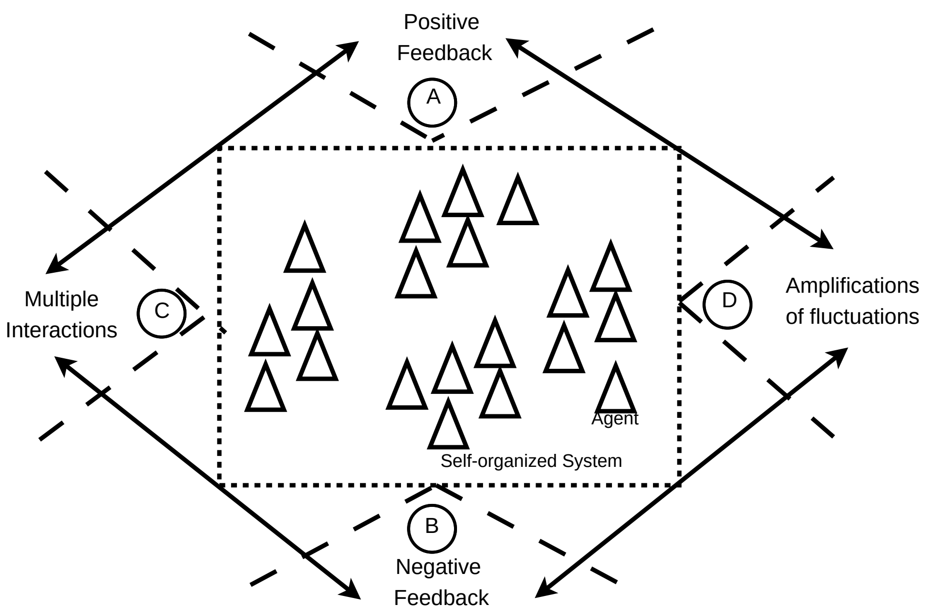


Figure 2.2: Self-organization viewed from four (A-D) inseparable perspectives. Adopted from **?**.

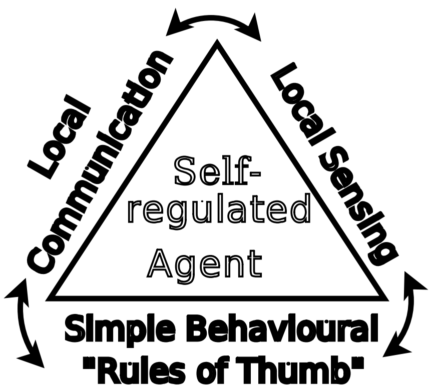


Figure 2.3: Three major interfaces of a self-regulated agent.

(a) Honey-bee nest (b) Construction of honey-combs

Figure 2.4: (a) A Honey-bee colony has built a nest on a tree-branch. From http://www.harunyahya.com, last seen on 01/05/2010. (b) Honey-bees are constructing honey combs. From http://knol.google.com, last seen on 01/05/2010.

dered homing of bats (Fig. 2.6). Since the discovery of these collective behavioural patterns of self-organized societies, scientists observed modulation or adaptation of behaviours in the individual level (**?**). For example, in order to prevent a life-threatening humidity-drop in the colony, cockroaches maintain a locally sustainable humidity level by increasing their tendency to aggregate, i.e. by regulating their individual aggregation behaviours . As shown in Fig. 2.3, this self-regulation (SR) of an individual agent is depicted through a triangle where its base-arm of simple behavioural rules of thumb (e.g. in this case, intense aggregation of cockroaches in low humidity) is supported by two side-arms: local communication and local sensing. This local sensing is sometimes also referred to as sensing or information gathering from the work in progress, e.g.stigmergy, and the local communication mentioned here is an instance of direct communication with neighbours (**?**).

SR has been studied in many other branches of knowledge. In most places of literature, SR refers to the exercise of control over oneself to bring the self into line with preferred standards (**?**). One of the most notable self-regulatory process is the human body's homoeostatic process where the human body's inner process seeks to return to its regular temperature when it gets overheated or chilled. **?** has referred self-regulation to goal-directed behaviour or feedback loops, whereas self-control may be associated with conscious impulse control. In psychology, SR denotes the strenuous actions to resist temptation or to overcome anxiety. SR is also divided into two categories: 1) conscious and 2) unconscious SR. Conscious

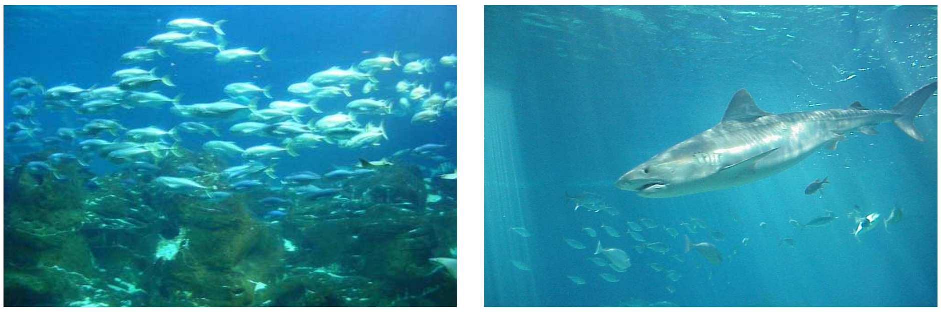

(a) A moving school of fish (b) A shark attacked a school of fish

Figure 2.5: (a) Group cohesion of a school of fish. (b) When a shark attacks a school of fish, they misguide the attacker by swift random movements. From http://www.travelblog.org last seen on 01/05/2010.

SR puts emphasis on conscious, deliberate efforts in self-regulation. On the other hand, unconscious self-regulation refers to the automatic self-regulatory process that is not labour intensive but operate in harmony with unpredictable, unfolding events in the environment. This uses the available informational input in ways that help to attain an activated goal.

The concepts of SR is also commonly used in cybernetic theory where SR in inan-

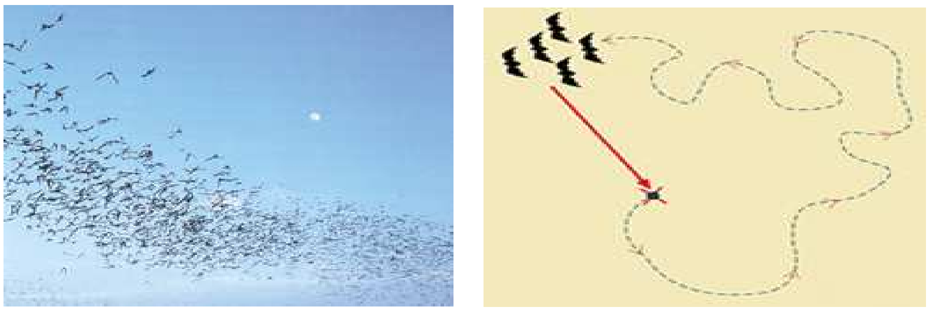

(a) A bat colony (b) Navigation of bats

Figure 2.6: (a) One of the largest bat colony with about 50 million bats, (b) These bats can show amazing navigation abilities: they always fly back to their nest on a straight route from wherever they are, reproduced from **?**.

imate mechanisms show that they can regulate themselves by making adjustments according to pre-programmed goals or set standards. A common example of this kind can be found in a thermostat that controls a heating and cooling system to maintain a desired temperature in a room. In physics, chemistry, biology and some other branches of natural sciences, the concept of SR is centred around the study of self-organizing individuals. SR has also been studied in the context of human social systems where it originates from the division of social labour that creates self-organized process that has self-regulating effects (**?**). Two types of SR have been reported in many places of literature of sociology: 1) SR from SO and 2) SR from activities of components in a heterarchical organization. It is interesting to note that SR in biological species provides the similar evidences of bottom-up approach of SR of heterarchical organization through interaction of individuals or the absence of strict hierarchy (**?**).

From the above discussion, we see that the term *self-regulation* carries a wide range of meaning in different branches of knowledge. In psychology and cognitive neuroscience, SR is discussed within an individual's perspective whereas, in biology and social sciences, SR is discussed within the context of a group of individuals or society as a whole. In this thesis, the latter context is more appropriate where SR covers both aspects of monitoring one's own state and environmental changes in relation to the communal goal and thus making adjustments in self behaviours with respect to the changes found.

2.1.2 Communication

What is Communication ? Defining *communication* can be challenging due to the use of this term in several disciplines with somewhat different meanings. This has been potrayed in the writing of Sarah Trenholm (**?**) who describes communication as piece of luggage overstuffed with all manner of odd ideas and meanings. **?** defines communication as:

> "A social process where individuals employ symbols to establish and interpret meaning in their environment."

The notion of being a "social process" involves, at least two or more individuals, and interactions that are both dynamic and ongoing. Moreover, symbols can be simply some sort of arbitrary labels given to a phenomena and they can represent concrete objects or an abstract ideas. Encyclopaedia Britannica also defines communication as "the exchange of meanings between individuals through a common system of symbol". But since this definition lacks the notion of sociality we find it incomplete. There are many other debates related to communication, e.g. the

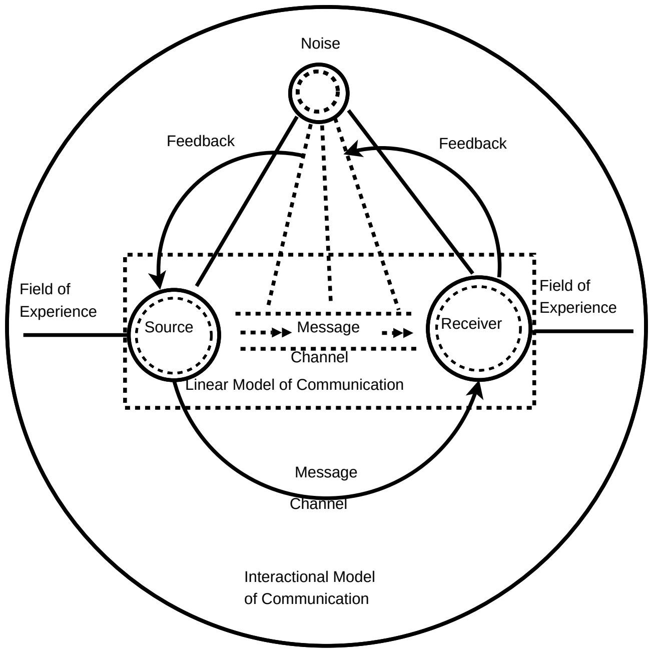


Figure 2.7: General models of communication, adopted from **?**.

intentionality debate (**?**), symbol grounding (**?**) and so on. However, in order to draw some tractable boundaries , we consider communication process within the context of symbol or message exchange between two or more parties with a clear intent to influence each others' behaviours.

According to a biological model of communication (Fig. 2.8), communication is a biological process where an individual (sender) intentionally transmits encoded message though physical signal and that, on being received and decoded by another individual of same species (receiver), influences receiver's behaviour (**?**). Note that, here individuals are of same species and thus they have a shared message vocabulary and mechanism of message encoding/decoding. Although this definition has not included the dynamics of a communication process, it is more precise for low-level biological and artificial systems. It accounts for the behavioural changes during communication process. These changes can be tracked though observing states of individuals.

Models of Communication. If we explore the elements of communication, we can grab the whole picture involved in the communication process. This can be explained through the study of the models of communication. There exists a plenty of

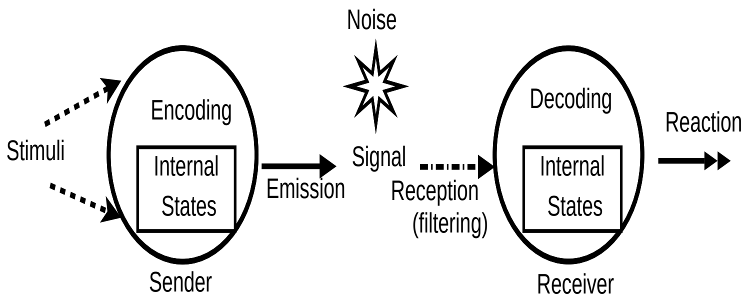


Figure 2.8: A biological model of communication, adopted from **?**.

models of communication. For this thesis, here we briefly discuss three prominent models: 1) linear model, 2) interaction model and 3) transaction model. Fig. 2.7 embeds first two models inside a single circle. In linear model, as introduced by Claude Shanon and Warren Weaver in 1949, communication is a one way process where a *message* is sent from a *source* to a *receiver* through a *channel*. In Fig. **??**, around this linear view, interaction model has been drawn. This model, proposed by Wilber Schramm in 1954, views communication as a two-way process with an additional *feedback* element that links both source and receiver. This feedback is a response given to the source by the receiver to confirm how the message is being understood. Here, during message passing, both source and receiver utilize their individual *field of experiences* that describe the overlap of their common experiences, cultures etc. Unlike separate filed of experiences and discrete sending and receiving of message, in transactional model, introduced by Barnlund in 1970, the sending and receiving of message is done simultaneously. Here, the field of experiences of source and receiver can overlap to some degrees. In all of the above three models, *noise* or a common message distorting element is present in the communication process. This noise can be occurred from the linguistic influences, i.e. message semantics, physical or bodily influences, cognitive influences or even from biological or physiological influences e.g., anger or shouting voice while talking.

The above models of communication describe the incremental complexities of message exchanging in the communication process. Surely, the transactional model is comparatively the most sophisticated model that prescribes adjusting the sender's message content while receiving an implicit or explicit feedback in real-time. For example, while speaking with her son for advising to read a story book, a mother may alter her verbal message as she simultaneously "reads" the non-verbal message of her child from his face. However, in case of a MRS, such sophistication may not be required or realizable by the current state of the art in multi-robot communication technology. In this study, we follow the simple linear model that meets the most of communication requirements of a MRS. The feedback has not been considered as we have assumed that all robots of our artificial MRS have a same shared vocabulary such that a message is understood as it is sent. Source never waits for an additional feedback from receiver to terminate sending a message.

Measuring communicated information. Following the linear model of communication, the amount of transferred information associated with a certain random variable X can be calculated by the concept of *Shanon entropy*. Adopting the notation of Feldman (**?**), and indicating a discrete random variable with the capital letter X, that can take values $x \in \chi$, the information entropy is defined as:

$$H[X] = -\sum_{x \in \chi} p(x) \cdot \log_2 p(x) \tag{2.1}$$

where $p(x)$ is the probability that X will take the value of x. $H[X]$ is also called the *marginal* entropy of X , since it depends on only the marginal probability of one random variable. The marginal entropy of the random variable X is zero if X always assumes the same value with $p(X = x) = 1$, and maximum if X assumes all possible states with an equal probability.

For example, in order to measure information flow in an elementary communication system, let *bit* be the unit amount of information needed to make a choice between two equiprobable alternates. If n alternates are present, a choice provides the following quantity of information: $H = log_2\, n$. Thus sending of n equiprobable messages reduces $log_2\, n$ amount of uncertainty and thus the amount of information is $log_2\, n$ bit. Similarly, according to Eq. 2.1, the value of $H[X]$ depends on the discretization of x. For instance, if the value of random variable x is discretized into 4, then $p(x)$ becomes $\frac{1}{4}$ leading to $H[X]$ = - 4 · $\frac{1}{4}$ · $log_2\, \frac{1}{4}$ = 2.

Communication modes and strategies. The communication structure of a system can broadly be classified into two major categories: centralized communication mode (CCM) and decentralized or local communication mode (LCM). A centralized communication system generally has a central entity, e.g. gateway, that

Table 2.1: General characteristics of common communication modes

Type	**Indirect or implicit communication**	**Direct or explicit communication**
Centralized Communication Mode (CCM)	Typically a central entity modifies the environment. It facilitates passive forms of communications, i.e. communication without specific target recipient.	Both global and local broadcast communications are commonly used. P2P communication can also occur. Here, exchange of messages occurs through a central entity.
Decentralized or Local Communication Mode (LCM)	All individuals are free to modify the environment and convey information to others.	P2P and local-broadcast are most commonly used forms. Global broadcast occurs to handle emergency situations. All communications are local without requiring a central entity.

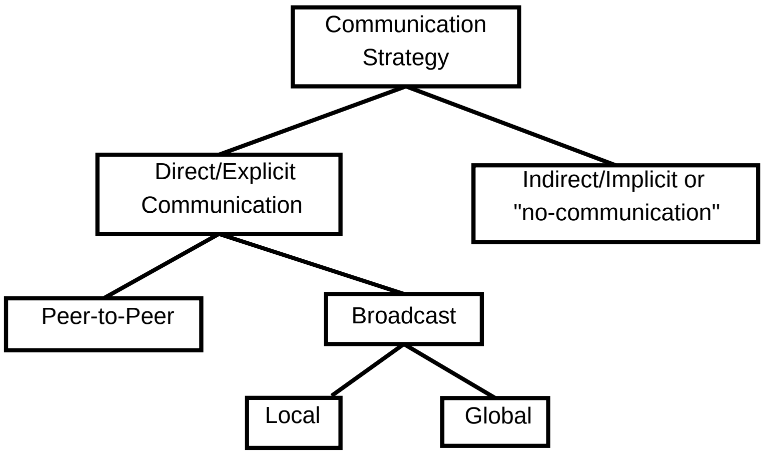

Communication Strategy
Direct/Explicit Communication
Indirect/Implicit or "no-communication"
Peer-to-Peer
Broadcast
Local
Global

Figure 2.9: Common communication strategies observed in social systems.

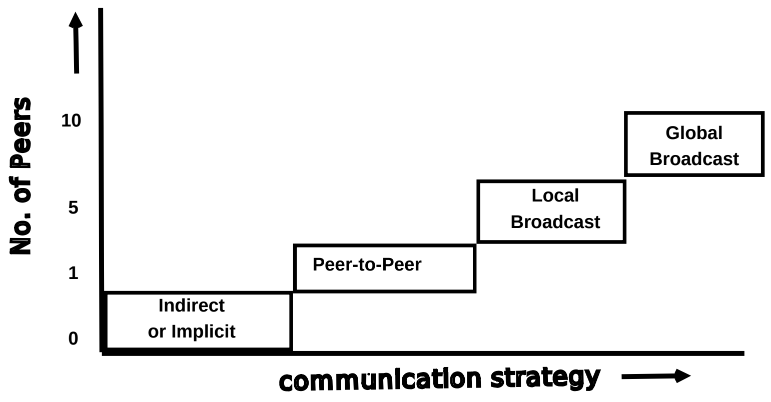


Figure 2.10: Number of recipients involved in various communication strategies.

routes all incoming and outgoing communications of the system. Individual nodes of this system often do not communicate each other directly. But they can send and receive messages through this central gateway. Central gateway can play many roles such as, access control, resource allocation and so on. On the other hand, in LCM, there is no central entity and each node can independently route message to each other.

In both biological and robotic literature two basic types of communication are often discussed: 1) direct or explicit communication and 2) indirect or implicit communication. As defined in (**?**), *direct communication* is an intentional communicative act of message passing that aims at one or more particular receiver(s). It typically exchanges information through physical signals. In contrast, *indirect communication*, sometimes termed as *stigmergy* in biological literature, happens as a form of modifying the environment, e.g. pheromone dropping by ants (**?**). In an ordinary sense, this is an observed behaviour and many robotic researchers call it as *no communication* (e.g. **?**). In order to avoid ambiguity, in this dissertation, by the term *communication*, we always refer to *direct* communication.

Direct or explicit communication can be limited by a communication range and thus by a number of target recipients. Under both CCM and LCM, nodes can select a certain number of target recipients of their messages. This process specifies *to whom* a node intends to communicate. In this thesis, we have denoted this mechanism of target recipient(s) selection as *communication strategy*. Fig. 2.9 shows the most common communication strategies found in a social system. In the simplest case, when only two nodes can communicate we call this *peer-to-peer* (P2P) communication. When nodes can spread information to a limited number of peers of their locality, the communication takes the form of *local broadcast*, i.e. one sender and a few receivers within a certain locality. For example, when a foraging honey-bee gives the information of flower sources to a number of peers through various dances, it conveys this information to a few peers through a local broadcast. However, giving the sample of nectar through tactile or taste to its peers can be considered as an instance of P2P communication. The *global broadcast* strategy can be found in almost all social species to handle emergency situations, e.g. emitting alarm signal in danger. Table 2.1 shows the relationship between various communication modes and their ways of adopting different strategies. Fig. 2.10 shows a typical count of average number of peers in various communication strategies. The actual number of peers under local broadcast strategy is dependent upon a particular social system and it changes over time in different levels of interactions among individuals. Sec. 2.2 and 2.5 reviews communication in biological social system (BSS) and MRS respectively.

2.1.3 Division of labour or task-allocation

Encyclopaedia Britannica serves the definition of *division of labour (DOL)* as the "separation of a work process into a number of tasks, with each task performed by a separate person or group of persons". Originated from economics and sociology, the term division of labour is widely used in many branches of knowledge. As mentioned by the Scottish philosopher Adam Smith, the founder of modern economics :

> The great increase of the quantity of work which, in consequence of the division of labour, the same number of people are capable of performing, is owing to three different circumstances; first, to increase the dexterity in every particular workman; secondly, to the saving of the time which is commonly lost in passing from one species of work to another; and lastly, to the invention of a great number of machines which facilitate and abridge labour, and enable one man to do the work of many.
> (Adam Smith (1776) in **?**)

In sociology, DOL usually denotes the work specialization (**?**). Basically, it answers three major questions:

1. *What task?* i.e., the description of the tasks to be done, service to be rendered or products to be manufactured.
2. *Why dividing it to individuals?* i.e., the underlying social standards for this division, such as task appropriateness based on class, gender, age, skill etc.
3. *How to divide it?* i.e.,the method or process of separating the whole task into small pieces of tasks that can be performed easily.

(a) A termite nest (b) Two Skyscrapers

Figure 2.11: (a) A termite colony constructs their nest through bottom-up approach, i.e. without a central planner. (b) Humans construct skyscrapers using a top-down plan. From http://www.harunyahya.com, last seen on 01/05/2010.

From the study of biological social insects and other BSSes, we can find that two major metrics of DOL have been established in literature: 1) task-specialization and 2) plasticity. *Task-specialization* is an integral part of DOL where a worker usually does not perform all tasks, but rather specializes in a set of tasks, according to its morphology, age, or chance (**?**). This DOL among nest-mates, whereby different activities are performed simultaneously by groups of specialized individuals, is believed to be more efficient than if tasks were performed sequentially by unspecialised individuals. DOL also has a great *plasticity* where the removal of one class of workers is quickly compensated for by other workers. Thus distributions of workers among different concurrent tasks keep changing according to the external (environmental) and internal conditions of a colony (**?**).
In artificial social systems, like multi-agent or MRS, the term “division of labour”

is often found synonymous to "task-allocation" (**?**). However, some researchers (e.g. (**?**)) argued to distinguish these terms due to the origin and particular contextual use of these terms. Particularly, DOL adopts the biological notion of collective task performance with little or no communication. On the other hand, task allocation follows the meaning of assigning task(s) to particular robot(s) based on individual robot capabilities, typically through explicit communication, such as *intentional cooperation* (**?**). Generally, the former is considered by *swarm robotic system (SRS)* and latter is done under *traditional MRS*. Sec. 2.3.1 covers both of these approaches and Sec. 2.4 provides critical review on DOL under these approaches.
In this dissertation, for defining DOL we closely follow the SR approach that emphasizes on having task-allocation and plasticity among workers. However, we do not put any restriction on the use of communication. In fact, we view DOL as a group-level phenomenon which occur due to the individual agent's self-regulatory task selection behaviour. But, unlike SR approach that view communication as expensive and hence try to find solutions avoiding it, we do not advocate for restricting the use of communication. Rather, along with our generic mechanism of division of labour, i.e. AFM (Chapter 4), we have proposed some self-regulatory communication strategies to vary communication load dynamically (Chapter 5).

2.2 Communication in biological social systems

Communication plays a central role in self-regulated division of labour of biological social systems.In this section, communication among biological social insects are briefly reviewed within the context of self-regulated division of labour.

2.2.1 Purposes, modalities and ranges

Communication in biological societies serves many closely related social purposes. Most peer-to-peer (P2P) communication include: recruitment to a new food source or nest site, exchange of food particles, recognition of individuals, simple attraction, grooming, sexual communication etc. In addition to that colony-level broadcast communication include: alarm signal, territorial and home range signals and nest markers, communication for achieving certain group effect such as, facilitat-

Table 2.2: Common communication modalities in biological social systems

Modality	**Range**	**Information type**
Sound	Long[a]	Advertising about food source, danger etc.
Vision	Short[b]	Private, e.g. courtship display
Chemical	Short/long	Various messages, e.g. food location, alarm etc.
Tactile	Short	Qualitative info, e.g. quality of flower, peer identification etc.
Electric	Short/long	Mostly advertising types, e.g. aggression messages

[a] Depending on the type of species, long range signals can reach from a few metres to several kilometres.
[b] Short range typically covers from few mm to about a metre or so.

ing or inhibiting a group activity (**?**).

Biological social insects use different modalities to establish social communication, such as, sound, vision, chemical, tactile, electric and so forth. Sound waves can travel a long distance and thus they are suitable for advertising signals. They are also best for transmitting complicated information quickly (**?**). Visual signals can travel more rapidly than sound but they are limited by the physical size or line of sight of an animal. They also do not travel around obstacles. Thus they are suitable for short-distance private signals such as in courtship display.

In ants and some other social insects chemical communication is dominant. Any kind of chemical substance that is used for communication between intra-species or inter-species is termed as semiochemical (**?**). A pheromone is a semiochemical, usually a glandular secretion, used for communication within species. One individual releases it as a signal and others responds it after tasting or smelling it. Using pheromones individuals can code quite complicated messages in smells. For example a typical an ant colony operates with somewhere between 10 and 20 kinds of signals (**?**). Most of these are chemical in nature. If wind and other conditions are favourable, this type of signals emitted by such a tiny species can be detected from several kilometres away. Thus chemical signals are extremely economical of their production and transmission. But they are quite slow to diffuse away. But ants and other social insects manage to create sequential and compound messages either by a graded reaction of different concentrations of same substance or by blends of signals. Tactile communication is also widely observed in ants and other species typically by using their body antennae and forelegs. It is observed that in ants touch is primarily used for receiving information rather than informing something. It is usually found as an invitation behaviour in worker recruitment process. When an ant intends to recruit a nest-mate for foraging or other tasks it runs upto a nest-mate and beats her body very lightly with antennae and forelegs. The recruiter then runs to a recently laid pheromone trail or lays a new one. In this form of communication limited amount of information is exchanged. In underwater environment some fishes and other species also communicate through electric signals where there nerves and muscles work as batteries. They use continuous or intermittent pulses with different frequencies learn about environment and to convey their identity and aggression messages.

(a) Flashing fireflies

(b) A firefly emitting light

Figure 2.12: (a) Flashing lights of fireflies displaying their synchronous behaviours (b) A firefly can produce light to signal other fireflies. From http://www.letsjapan.markmode.com, last seen on 01/06/2010.

2.2.2 Signal active space and locality

The concept of active space is widely used to describe the propagation of signals by species. In a network environment of signal emitters and receivers, active space is defined as the area encompassed by the signal during the course of transmission (**?**). In case of long-range signals, or even in case of short-range signals, this area include several individuals where their social grouping allows them to stay in cohesion. The concept of active space is described somewhat differently in case some social insects. In case of ants, this active space is defined as a zone within which the concentration of pheromone (or any other behaviourally active chemical substances) is at or above threshold concentration (**?**). Mathematically this is denoted by a ratio:

$$AS = \frac{\textit{Amount of pheromone emitted (Q)}}{\textit{Threshold concentration at which the receiving ant responds (K)}} \qquad (2.2)$$

Q is measured in number of molecules released in a burst or in per unit of time whereas K is measured in molecules per unit of volume. Fig. 2.13 shows the use of active spaces of two species of ants: (a) *Atta texana* and (b) *Myrmicaria eumenoides*. The former one uses two different concentrations of *4-methyl-3-heptanone* to create attraction and alarm signals while the latter one uses two different chemicals: *Beta-pinene* and *Limonene* two create similar kinds signals, i.e. alerting and circling.

The adjustment of this ratio enables individuals to gain a shorter fade-out time and permits signals to be more sharply pinpointed in time and space by the receivers. In order to transmit the location of the animal in the signal, the rate of information transfer can be increased by either by lowering the rate of emission of Q or by increasing K, or both. For alarm and trail systems a lower value of this ratio is used. Thus, according to need, individuals regulate their active space by making it large or small, or by reaching their maximum radius quickly or slowly, or by enduring briefly or for a long period of time. For example, in case of alarm, recruitment or sexual communication signals where encoding the location of an individual is needed, the information in each signal increases as the logarithm of the square of distance over which the signal travels. From the precise study of pheromones it has been found that active space of alarm signal is consists of a concentric pair of hemispheres (see Fig. 2.13). As the ant enters the outer zone she is attracted inward toward the point source; when she next crosses into the central hemisphere she become alarmed. It is also observed that ants can release pheromones with different active spaces.

Active space has strong role in modulating the behaviours of ants. For example, when workers of *Acanthomyops claviger* ants produce alarm signal due to an attack by a rival or insect predator, workers sitting a few millimetres away begin to react within seconds. However those ants sitting a few centimetres away take a minute or longer to react. In many cases ants and other social insects exhibit modulatory communication within their active space where many individuals involve

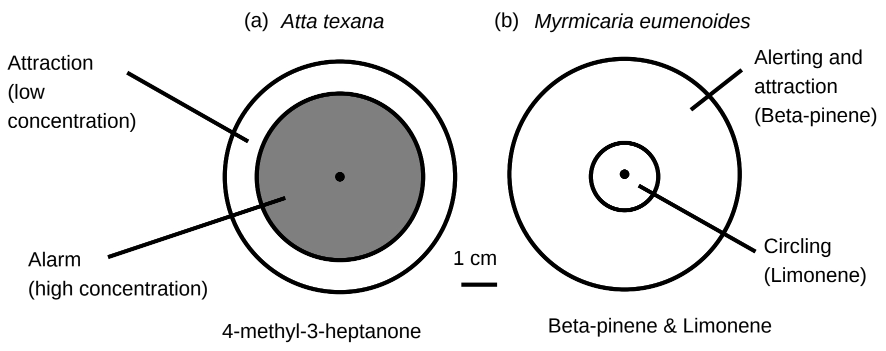

(a) *Atta texana*
(b) *Myrmicaria eumenoides*
Attraction (low concentration)
Alarm (high concentration)
1 cm
4-methyl-3-heptanone
Alerting and attraction (Beta-pinene)
Circling (Limonene)
Beta-pinene & Limonene

Figure 2.13: Pheromone active space observed in ants, reproduced from **?**.

in many different tasks. For example, while retrieving the large prey, workers of *Aphaeonogerter* ants produce chirping sounds (known as stridulate) along with releasing poison gland pheromones. These sounds attract more workers and keep them within the vicinity of the dead prey to protect it from their competitors. This communication amplification behaviour can increase the active space to a maximum distance of 2 meters.

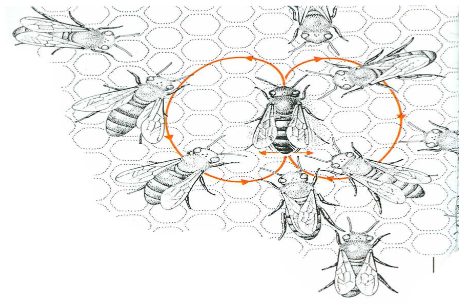
2.2.3 Common communication strategies

In biological social systems, we can find all different sorts of communication strategies ranging from indirect pheromone trail laying to local and global broadcast of various signals. Sec. 2.1.2 discusses the most common four communication strategies in natural and artificial world, i.e. indirect, P2P, local and global broadcast communication strategies. Table 2.3 lists the use of various communication modalities under different communication strategies. Here we give a few real examples of those strategies from biological social systems. In biological literature, the pheromone trail laying is one of the most discussed indirect communication strategy among various species of ants. Fig. 2.14 shows a pheromone trail following of a group of foraging ants. This indirect communication strategy effectively helps ants to find a better food source among multiple sources, find shorter distance to a food source, marking nest site and move there etc. (**?**).

Direct P2P communication strategy is also very common among most of the biological species. Fig. **??** and Fig. **??** shows P2P communication of ants and honey-bees respectively. This tactile form of communication is very effective to

Figure 2.14: A group of ants following pheromone-trail. From http://www.bioteams.com, last seen on 01/06/2010.

Figure 2.15: A dancing honey-bee (*centre*) and its followers. From http://knol.google.com, last seen on 01/06/2010.


(a) Honey-bee's waggle dance

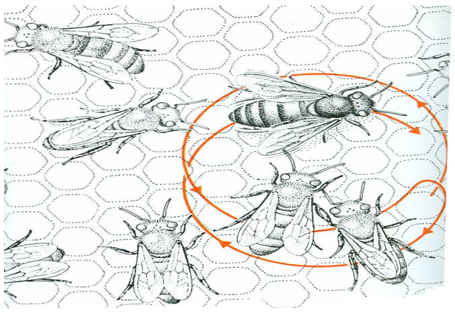

(b) Honey-bee's round dance

Figure 2.16: Examples of local broadcast communication of honey-bees: (a) Honey-bees show waggle-dance (making figure of 8) when food is far and (b) they show round-dance without any waggle when food is closer (within about 75m of hive). From **?**.

exchange food item, flower nectar with each-other or this can be useful even in recruiting nest-mates to a new food source or nest-site.

2.2.4 Roles of communication in task-allocation

Communication is an integral part of the DOL process in biological social systems. It creates necessary preconditions for switching from one tasks to another or to attend dynamic urgent tasks. Suitable communication strategies favour individuals to select a better tasks. For example, **?** has reported two worker-recruitment experiments on black garden ants and honey-bees. The scout ants of *Lasius niger* recruit

Table 2.3: Common communication strategies in biological social systems

Communication strategy	**Common modalities used**
Indirect	Chemical and electric
Peer-to-peer (P2P)	Vision and tactile
Local broadcast	Sound, chemical and vision
Global broadcast	Sound, chemical and electric

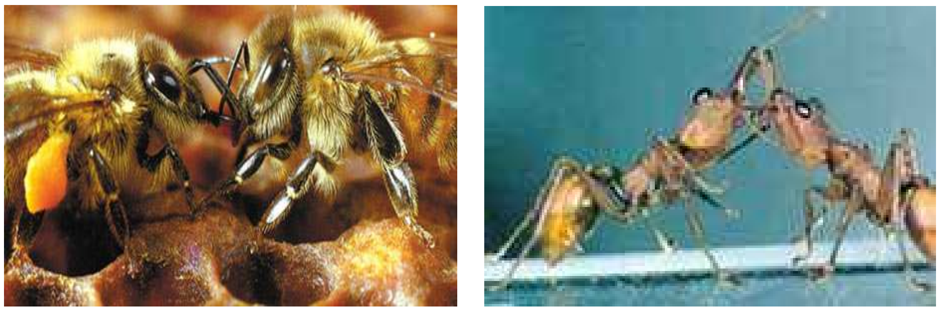

(a) Two honey-bees (b) Two ants

Figure 2.17: Example of P2P tactile communication: (a) Honey-bees exchange nectar samples by close contact (b) ants also exchange food or information via tactile communication From http://www.harunyahya.com/ last seen 01/05/2010 .

Table 2.4: Self-regulation of communication behaviours based on task-urgency perception

Example event	**Strategy**	**Modulation of communication based-on task-urgency**
Ant’s alarm signal by pheromones	Global broadcast	High concentration of pheromones increase aggressive alarm-behaviours
Honey-bee’s round dance	Local broadcast	High quality of nectar source increases dancing and foraging bees
Ant’s tandem run for nest selection	P2P	High quality of nest increases traffic flow
Ant’s pheromone trail-laying to food sources	Indirect	Food source located at shorter distance gets higher priority as less pheromone evaporates and more ants joins

uninformed ants to food source using a well-laid pheromone trails. *Apis mellifera* honey-bees also recruit nest-mates to newly discovered distant flower sources through waggle-dances. In the experiments, poor food sources are given first to both ants and honey-bees. After some time, rich food has been introduced to both of them. It has been found that only honey-bees can switch from poor source to a rich source. The sophisticated dance communication of honey-bees favours them to get a better solution.

Table 2.4 presents the link between the task-urgency perception and self-regulation of communication behaviours in biological social systems. Here, we can see that communication is modulated based on the perception of task-urgency irrespective of the communication strategy of a particular species. This dissertation takes this biological evidence as the baseline of our hypothesis that communication behaviours of a self-regulated system must be linked to the task-requirements of the society. Under indirect communication strategy of ants, i.e. pheromone trail-laying, we can see the that principles of self-organization, e.g. positive and negative feedbacks take place due to the presence of different amount of pheromones for different time periods. Initially, food source located at shorter distance gets relatively more ants as the ants take less time to return nest. So, more pheromone deposits can be found in this path as a result of positive feedback process. Thus, the density of pheromones or the strength of indirect communication link reinforces ants to

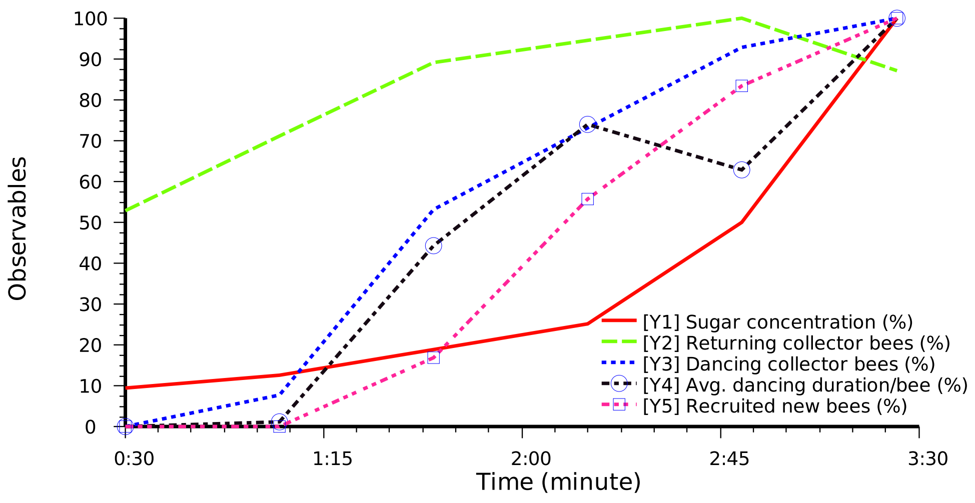


Figure 2.18: Self-regulation in honey-bee's dance communication behaviours, produced after the results of **?** honey-bee round-dance experiment performed on 24 August 1962.

follow this particular trail.

Similarly, perception of task-urgency influences the P2P and broadcast communication strategies. *Leptothorax albipennis* ant take lees time in assessing a relatively better nest site and quickly return home to recruit its nest-mates (**?**). Here, the quality of nest directly influences its intent to make more "tandem-runs" or to do tactile communication with nest-mates. We have already discussed about the influences of the quality of flower sources to honey-bee dance. Fig. 2.18 shows this phenomena more vividly. It has been plotted using the data from the honey-bee round-dance experiments of **?**. In this plot, Y1 line refers to the concentration of sugar solution. This solution was kept in a bowl to attract honey-bees and the amount of this solution was varied from $\frac{3}{16}$M to 2M (taken as 100%). The variation of this control parameter influenced honey-bees communication behaviours and thus they produce varying degree of division of labour. Y2 line in Fig. 2.18 represents the number of collector bees that return home. The total number of collectors was 55 (taken as 100%). Y3 plots the percent of collectors displaying round dances. We can see this dancing collectors is directly proportional to the concentration of sugar solution or task-urgency in this case. Similarly the average duration of dance per bee is plotted in Y4 line. The maximum dancing period was 23.8s (taken as 100%). Finally, from Y5 line we can see the outcome of the round-dance communication as the number of newly recruited bees to the feeding place. The maximum number of recruited bees was 18 (taken as 100%). So, from an overall observation, we can see that bees sense the concentration of food-source as the task-urgency and they self-regulate their round-dance behaviour according to this task-urgency. Thus, this self-regulated dancing behaviour of honey-bees attracts an optimal number of inactive bees to work.

Broadcast communication is one of the classic way to handle dynamic urgent tasks in biological social systems. It can be commonly observed in birds, ants, bees and many other species. Table 2.4 mentions about the alarm communication of ants. Similar to the honey-bee's dance communication, ants has a rich language of chemical communication that can produce words through blending different glandular secretions in different concentrations. Fig. 2.13 shows how ants can use different concentrations of chemicals to make different stimulus for other ants. From the study of ants, it is clear to us that taking defensive actions, upon sensing a danger, is one of the highest-priority tasks in an ant colony. Thus, in this highly urgent task, ants almost always use their global broadcast communication strategy through their strong chemical signals and they make sure all individuals can join in this task. This gives us a coherent picture of self-regulation of animal communication based on the perception of task urgency.

2.2.5 Effect of group size on communication

The performance of cooperative tasks in large group of individuals also depends on communication and information sharing. From the study of social wasps, **?** has reported that, depending on the group size, different kinds of information flow occur in two groups of social wasps: 1) independent founders of *Polistes* (Fig. **??**) 2) swarm founders of *Polybia* (Fig. **??**). Independent founders (IF) are species in which reproductive females establish colonies alone or in small groups, but independent of any sterile workers and in the range of 10^2 individuals at maturity. Swarm founders (SF) initiates colonies by swarm of workers and queens. They have a large number of individuals, in the order of 10^6 and 20% of them can be queen. The most notable difference in the organization of work of these two social wasps is IF does not rely on any cooperative task performance while SF interact with each-other locally to accomplish their tasks. The work mode of IF can be

(a) Polybia wasps

(b) Polistes wasps

Figure 2.19: Colony founding in two types of social wasps (a) *Polybia occidentalis* founds colony by swarms (b) *Polistes* founds colony by a few queens independently. From http://www.discoverlife.org, last seen 01/05/2010.

considered as *global sensing no communication (GSNC)* where individuals sense the task-urgencies throughout a small colony and do these tasks without communicating with each other. On the other hand, the work mode of SF can be treated as *local sensing local communication (LSLC)* where individuals can only sense locally due to large colony-size and they communicate locally to exchange information, e.g. task-urgency (although their exact mechanism is unknown).

Fig. 2.20 compares the occurrence of information flow among IF and SF. In case of SF information about nest-construction or broods food-demand can not reach to foragers directly. Fig. 2.21 shows the path of information flow among SF for nest construction. The works of *pulp foragers* and *water foragers* depend largely on their communication with *builders*. On the other hand, in case of IF there is no such communication present among individuals. This phenomena raises the question of how these individuals can select the best work modes from GSNC and LSLC.

? tried to answer this question in terms of task-specialization. In case of large colonies, many individuals repeatedly performs same tasks as this minimizes their interferences, although they still have a little probability to select a different task randomly (**?**). But because of the large group size, the queuing delay in inter-task switching keeps this task-switching probability very low. Thus, in SF, task-specialization becomes very high among individuals. On the other hand, in small group of IF, specialization in specific tasks is costly because these prevents individuals not to do other tasks whose task-urgency becomes high. Thus they becomes

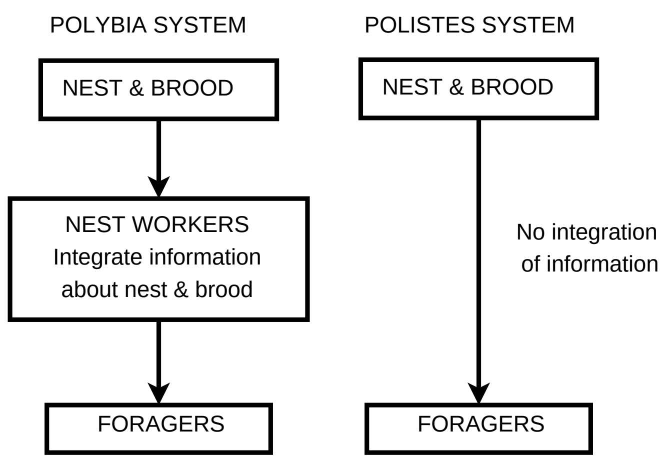


Figure 2.20: Different patterns of information flow in two types of social wasps: polybia and polistes, reproduced from **?**.

generalist and do not communicate task-urgency to each other.

The above interesting findings on GSNC and LSLC in social wasps have been

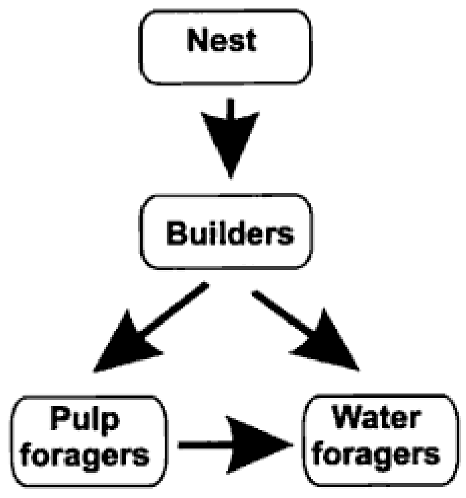


Figure 2.21: Information flow in polybia social wasps, reproduced from **?**.

linked up with the group productivity of wasps. Fig. 2.22 illustrates high group productivity in case of LSLC of SF. The per capita productivity was measured as the number of cells built in the nest in (a) and the weight of dry brood in grams in (b). In case of IF this productivity is much lesser (max. 24 cells per queen at the time the first offspring observed) comparing to the thousands of cells produced by SF (**?**). This shows us the direct link between high productivity of social wasps and their local communication and information strategy.

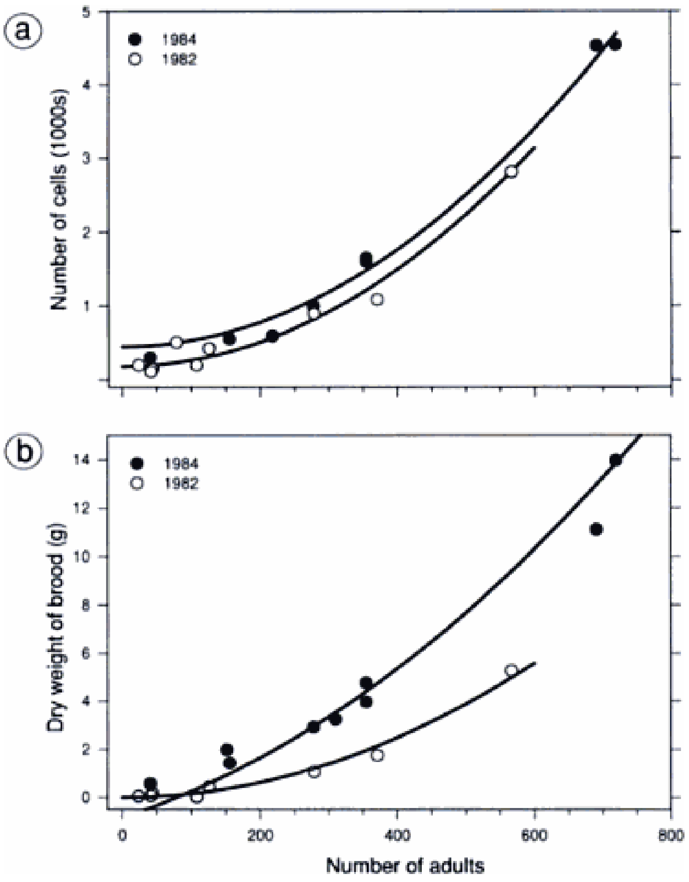


Figure 2.22: Productivity of social wasps as a function of group size, reproduced from **?**.

2.3 Overview of multi-robot systems (MRS)

2.3.1 MRS research paradigms

Historically the concept of multi-robot system comes almost after the introduction of behaviour-based robotics paradigm (**?**, **?**). In 1967, using the traditional sense-plan-act or hierarchical approach (**?**), the first Artificially Intelligent (AI) robot, Shakey, was created at the Stanford Research Institute. In late 80s, **?** influenced this entire field of mobile robotics by his layered, behaviour based robot-control approach that acted significantly differently than the hierarchical approach. At the same time, **?** described a set of experiments where increasingly complex vehicles

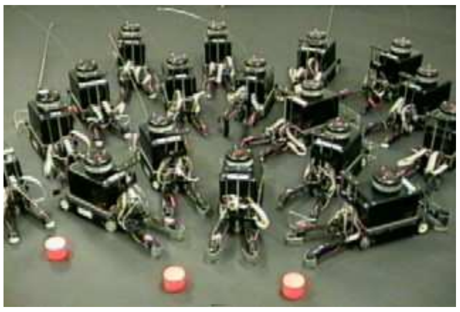


Figure 2.23: The Nerd-Herd. From **?**

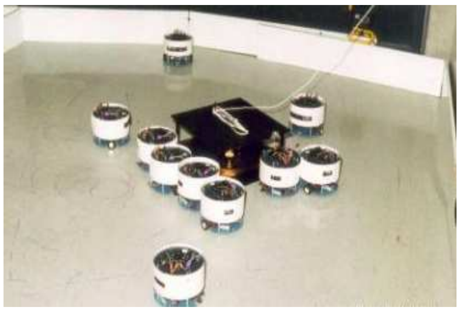


Figure 2.24: A group of 10 box pushing robots. From **?**

(a) A disaster site: buring oil rig

(b) An array of Seaglider underwater robots

Figure 2.25: Example of MRS working at a disaster site (a) British Petroleum's oil rig sinks in the Gulf of Mexico after explosion. From http:///news.bbc.co.uk, reported on 22/04/2010. (b) IRobot's Seaglider fleet that was surveying oil spill in the Gulf of Mexico at depths of up to 1,000 meters. From http:///www.irobot.com, reported on 25/05/2010.

are built from simple mechanical and electrical components. Around the same time and with similar principles, **?** developed a distributed behavioural model for a bird in a flock that assumed that a flock is simply the result of the interactions among the individual birds (see Sec. 2.3.4). Early research on multi-robot systems also include the concept of cellular robotic system (**?**, **?**) multi-robot motion planning (**?**, **?**, **?**) and architectures for multi-robot cooperation (**?**). Fig. 2.23 and 2.24 present us the two earliest MRS system in foraging and box-pushing task domains, developed by the pioneers in this filed, **?** and **?** respectively.

From the beginning of the behaviour based paradigm, the biological inspirations influenced many cooperative robotics researchers to examine the social characteristics of insects and animals and to apply them to the design multi-robot systems

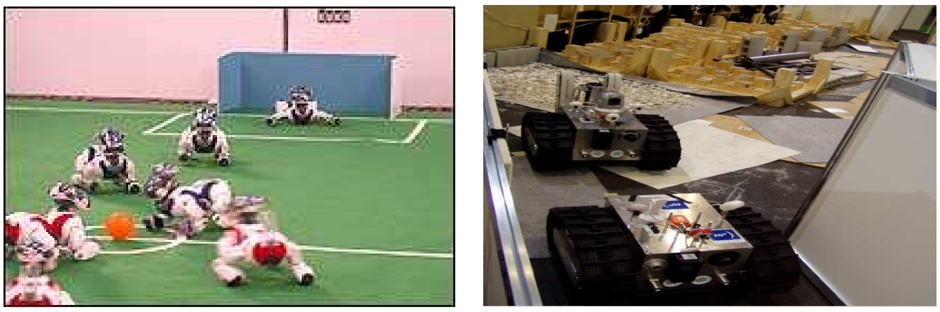

(a) Robot teams at Robocup soccer (b) Robot team at Robocup rescue

Figure 2.26: Example of MRS at sports and rescue (a) Robot dogs playing at Robocup soccer. From http:///news.bbc.co.uk, reported on 11/05/2005. (b)Rugbot robot team was competing at Osaka's Robocup rescue league from **?**.

(**?**). The underlying basic idea is to use the simple local control rules of various social species, such as ants, bees, birds etc., to the development of similar behaviours in multi-robot systems. In multi-robot literature, there are many examples that demonstrate the ability of multi-robot teams to aggregate, flock, forage, follow trails etc. (**?**, **?**). The dynamics of ecosystem, such as cooperation, has also been applied in multi-robot systems that has presented the emergent cooperation among team members (**?**, **?**). On the other hand, the study of competitive behaviours among animal and human societies has also been applied in multi-robot systems, such as that found in multi-robot soccer (**?**). From Fig. 2.25 and 2.26 show us that the applications of MRS can be ranging from human disaster recovery to games and entertainment.

As discussed above, there are several research groups who follow different approaches to handle multi-robot research problems. Based on the underlying philosophies and principles, We have classified them into two broad paradigms: 1) Traditional MRS and 2) Swarm-robotic system (SRS). Below we have highlighted them briefly.

1. Traditional MRS paradigm

As discussed above, unlike SRS, traditional MRS does not directly take inspiration from BSS. Rather it follows the organizational, social, knowledge-based and multi-agent based approaches to solve problems of MRS. Explicit modelling of environment, tasks, robots can be the main features of these systems. According to

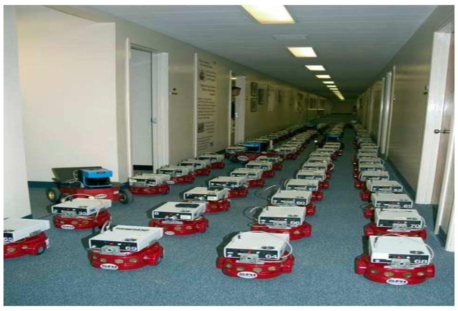


Figure 2.27: Hundreds of Centibots robots worked at indoor search, navigation and mapping tasks. From **?**.

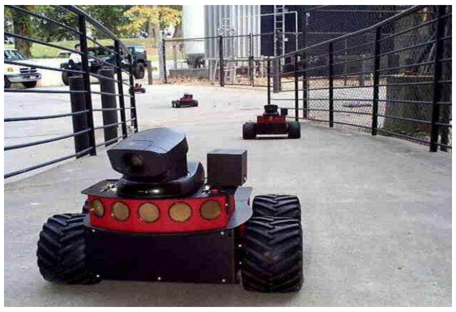


Figure 2.28: Pioneer robots operating at outdoor uncertain environment of Georgia Tech. Mobile Robot Lab. From http://news.cnet.com, reported on 05/04/2010.

?, traditional MRS can be classified into two following categories.

Organizational and social approaches: Organizational and social paradigms are typically based on organization theory derived from human systems that reflects the knowledge from sociology, economics, psychology and other related fields. To solve complex problems this paradigm usually follows the cooperative and collaborative forms of distributed intelligence. In multi-robot systems the example of this paradigm is found in two major formats: 1) the use of roles and value system and 2) market economics. In multi-robot applications under this paradigm, an easy division of labour is achieved by assigning roles depending on the skills and capabilities in individual team member. For example, in multi-robot soccer (**?**, **?**) positions played by different robots are usually considered as defined roles. On the contrary, in market economics approach (**?**, **?**) task allocation among multiple robots are done via market economics theory that enables the selection of robots for specific tasks according to their individual capabilities determined by a bidding process.

Knowledge-based and multi-agent based approaches: This paradigm, commonly used for developing multi-agent systems, is knowledge-based, ontological and semantic paradigm. Here knowledge is defined as ontology and shared among robots/agents from disparate sources. It reduces the communication overhead by utilizing the shared vocabulary and semantics. Due to low bandwidth, limited power, limited computation and noise and uncertainty in sensing/actuation, the use of this approach is usually restricted in multi-robot systems.

2. SRS paradigm

In bio-inspired, emergent swarms paradigm local sensing and local interaction forms the basis of collective behaviors of swarms of robots. Many researchers addressed the issues of local interaction, local communication (i.e., stigmergy) and other issues of this paradigm (**?**), (**?**). Today, this paradigm has been emerged as a sub-field of robotics called swarm robotics (**?**). This is a powerful paradigm for those applications that require performing shared common tasks over distributed workspace, redundancy or fault-tolerance without any complex interaction of entities. Some examples include flocking, herding, searching, chaining, formations, harvesting, deployment, coverage etc.
Although our approximate classification of MRS includes most of the research directions it is very hard to specifically categorize all diverse researches on multi-robot systems. However, most of the researchers select a suitable paradigm to abstract the problem from an specific perspective with common fundamental challenges of MRS discussed in the later sections.

2.3.2 MRS taxonomies

The vast amount of research in MRS makes it necessary to use well-established classifications or taxonomies in order to specify and design the target MRS. In Section 2.3.1 we see that the main-stream research in MRS can be classified into two distinct paradigms. However, these paradigms have certain assumptions, often unspecified or implicit, regarding the design of MRS hardware, software, communication and interaction etc. Thus, MRS taxonomies can be useful for many purposes, e.g. to avoid ambiguities in system specification by reducing the size and complexity of possible design spaces, and to use certain trade-off among various features for achieving overall system performance.
While earlier MRS taxonomies, e.g. one proposed by **?**, **?** discuss very fundamental design issues of MRS, recent taxonomies e.g. proposed by **?**, **?**, **?**, **?** etc. gives us the detail design choices for making useful system specifications. We classify these recent taxonomies into two groups: 1) generalized taxonomies and

2) specialized taxonomies. We consider taxonomies of **?** and **?** as generalized taxonomies since they can be used to specify almost all necessary features of a MRS. On the other hand, specialized taxonomies provide MRS specification with respect to particular system features, e.g., the taxonomy of **?** is only useful in a MRS with reinforcement learning, the taxonomy of **?** gives us the specification of tasks in a MRTA context. Other less common taxonomies e.g., one proposed by **?** or another proposed by **?** are centred around the coordination (weak or strong or none), communication (implicit or explicit or none), architecture (centralized or decentralized) etc. Here we briefly describe the major axes of leading taxonomies of MRS.

Dudek's generalized taxonomy of MRS

? provides seven main axes of MRS specification. We have regrouped them into the following three major areas.

1. Collective or group size, composition and reconfigurability: A MRS can be formed by one, two, multiple or effectively infinite number of autonomous robots. Composition refers to the homogeneity of the group members. Robots can be identical in both form and function (hardware and software), homogeneous (consisting of same physical hardware) or physically heterogeneous. Collective reconfigurability refers to the rate at which robots can spatially re-position themselves. It can be completely static, coordinated or dynamically arranged.

2. Communication range, topology and bandwidth : The maximum distance between two robots, required for effective communication, can be zero (i.e. they can not communicate directly) or infinite (i.e. all robots can communicate to any other robot) or in-between these two. Communication topology determines the style of addressing target peers e.g., through broadcast messaging, individual addressing by name or address or, following tree-like hierarchy or redundant graphs. Communication bandwidth provides the measure of costs associated with communication. This can be no cost (i.e. infinite bandwidth) or high cost (i.e. limited or no bandwidth) or something in between these two extreme cases.

3. Processing abilities: This refers to the software architectures that can be used for controllers of the robots. General models are finite state automata, a push-down automata, neural-networks or Turing machines (most common assumption).

Specialized taxonomies of Gerkey and Balch

With this general speculation of MRS we can specify the tasks or rewards in the system using any specialized taxonomies. For example, taxonomy of **?** can be helpful to understand the target domain of application of MRS. He defined three axes of possible tasks and robot capabilities of MRS:
Single-task robots (ST) *vs.* multi-task robots (MT): ST (MT) means a robot can perform one (multiple) tasks at a time.
Single-robot tasks (SR) *vs.* multi-robot tasks (MR): while under SR each task requires only one robot, under MR, multi-robots may be required.
Instantaneous assignment (IA) *vs.* time-executed assignment (TA): While IA refers a situation when planning for future task-allocations is not possible, under TA planning is possible.
? extended this taxonomies of tasks and applied to multi-robot learning cases. His taxonomy of reward include: source of reward (internal or external or both), rewarding time (immediate or delayed), continuity of reward (discrete or continuous), locality of reward (global or local or a combination of both) relation to performance (tied to performance or based-on intuitive state-value).

Parker's taxonomy of interaction

In addition to the above two classes of taxonomies, we present here **?** notion of *interaction* in a robot team in four levels: 1) collective, 2) cooperative 3) collaborative and 4) coordinative. Although **?** did not claim it to be a MRS taxonomy, we found this very much useful to describe the high-level relationships of individuals of the system. Moreover this removes the ambiguities among these overused terms and makes them precise for future use. While analysing the role and application of distributed intelligence on MRS, **?** presented an excellent classification of interactions of entities of MRS. As seen in Fig. **??** she viewed the interactions along three different axes:

1. the types of goals of entities (either shared goal such as, cleaning a floor, or, individual goal)

2. whether entities have awareness of others on the team (either aware such as, in cooperative transport, or, unaware such as, in a typical foraging)

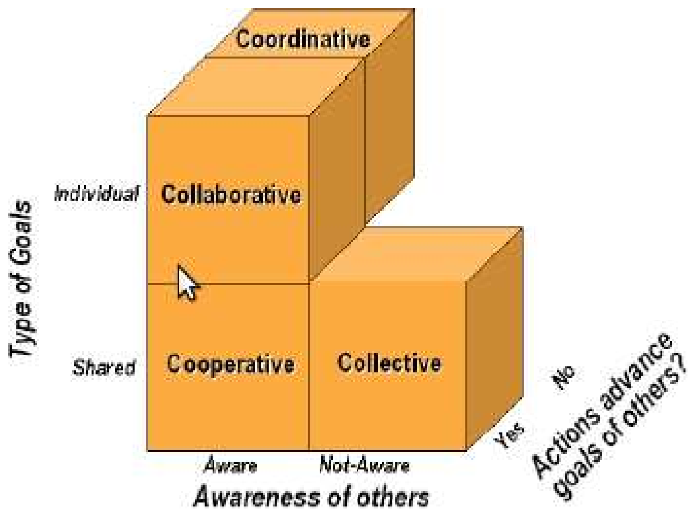


Figure 2.29: Categorization of types of interactions in MRS, reproduced from **?**.

3. whether the action of one entity advances the goal of others (e.g., one robot's floor cleaning helps other robots not to clean that part of the floor)

Based on this approximate observation Parker classified interactions into four categories:

1. Collective interaction: Entities are not aware of others on the team, yet do share goals and their actions are beneficial to team-mates. Mostly, swarm-robotic work of many researchers follow this kind of interaction to perform biologically-relevant tasks, such as foraging, swarming, formation keeping and so forth.

2. Cooperative interaction: Entities are aware of others on the team, they share goals and their actions are beneficial to their team-mates. This type of interaction is used to reason about team-mates capabilities multiple robots works together, usually in shared workspace, such as cleaning a work-site, pushing a box, performing search and rescue, extra-planetary exploration and so forth.

3. Collaborative interaction: Having individual goals (and even individual capabilities), entities aware of their team-mates and their actions are beneficial to their team-mates. One example of this kind of interaction is a team of collaborative robots where each must reach a unique goal position by sharing sensory capabilities to all members such as illustrated as coalition formation in (**?**).

4. Coordinative interaction: Entities are aware of each other, but they do not share a common goal and their actions are not helpful to other team members. For example, in a common workspace, robots try to minimize interference by coordinating their actions as found in multi-robot path planning techniques, traffic control techniques and so on.

Beyond this four most common types of interactions Parker also described another kind of interaction in adversarial domain where entities effectively work each other such as multi-robot soccer. Here entities have individual goals, they are aware of each other, but their actions have a negative affect on other robots' goals.

In this dissertation, we use the taxonomy of **?** for specifying our MRS (Sec. 3.1.3. The taxonomy of **?** is used to analyse the dependence of MRS on various levels of interactions in Sec. 2.4.

2.3.3 Traditional MRS

MRS not only shares the problem of controlling a single robot but also it amplifies the problem to several orders or magnitude. Below we list a few major challenges of any MRS:

Increased uncertainty about environment: When multiple robots work in a partially observable world, the environmental view becomes severely restricted due to both in terms of noisy sensor readings and frequent obstacle detections. Thus, in MRS, the uncertainty about the environment increased in may folds.

Increased dynamic changes of the environment: Since many robots work in a shared environment, the dynamic movements and physical interferences among the robots becomes more frequent and robots are required to change their course of action more frequently.

Decreased communication throughput: Interference in communication is inescapable for a team of robots. Since the typical bandwidth of a communication channel is fixed, adding more robots reduces the effective communication throughout and thus increased latency in robot-robot or robot-computer communications. If the robots are required to coordinate their action then the saturation of the communication channel affects the overall team-performance.

Decreased real-time performance: In a functional MRS, autonomous mobile robots need to do some tasks in real-time, e.g. identifying their current poses (*localization*) to determine next motions, avoiding obstacles etc. However, when the number of robots increases the real-time performance can be poor due to the above factors.

Increased sensor failures and break-downs: This is also common in a MRS that the real-time interaction of large number robots can decrease the life of their hardware as they become subject to more collisions and interferences. Thus overall reliability of the MRS can be decreased gradually.

Despite the above big challenges, robotic researchers design and operate MRS successfully using a number of intelligent solutions since the last few decades. In the previous subsections we have seen how the researches are classified into distinctive paradigms and can be specified by precise taxonomies. Here we list a number of typical issues that any MRS may encounter from its inception to implementation.

- *Motion control*. How to use sensor values to produce real-time motions avoiding obstacles ?
- *Localization*. How to find out the self-position in the world so that reaching to a specified target becomes possible ?
- *Navigation/map-building*. How to integrate sensor values to build maps or representation of the environment for further exploration ?
- *Task-selection*. How to plan/predict and select a particular high-level task (e.g. find a red object or picking up a stick) provided that a number of tasks present in the environment ?
- *Interaction and communication*.How to interact or communicate with other robots for cooperating, collaborating or coordination in doing tasks ?
- *Adaptation/learning* How to remember things so that future robot actions or behaviours become improved ?

Not all of the above issues are present in all MRS. Many MRSes do not use any form of navigation, or communication or learning and yet they do some useful tasks. However it is important to understand how these issues can be solved in a structured, modular and timely manner. Integrating the solutions of these issues and resolving the conflicts among them also appear to be the major functions of multi-robot control architectures. For example, conflicts occur if a resource is required by or, a unique single task is distributed to, more than one robot at any given

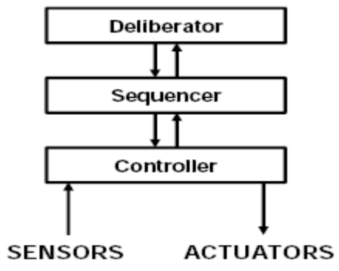


Figure 2.30: A typical hybrid robot control architecture, adopted from **?**.

time. Several resources such as bandwidth, space etc. may be needed by more than one robot (**?**). The sharing of bandwidth among robots is a great problem in case of applications like multi-robot mapping (**?**). As shown in Fig. **??**, in large multi-robot team such as in Centibots system (**?**), task interference and high bandwidth communication between 100s of robots appear as a significant research challenge. Whatever the principle characteristics of a MRS, e.g., homogeneity, coupling, communication methods etc., each MRS must address some degree to those problems. For example, usually every MRS adopts a control architecture under a specific paradigm to organize its hardware, software and communication system. Similarly every MRS address the issues of communication, localization, interaction in a way specific to the application and underlying design principles (or philosophies). In the following subsections, we have attempted to summarize the key MRS research issues that would influence the selection and implementation our research. In this initiative we have deliberately omitted the non-central or very specific issues, such as collaborative transport or reconfigurable MRS, that does not directly relate to our research.

Architecture and control

In MRS, two high-level control strategies are very common: 1) centralized and 2) decentralized or distributed. Under a specific control strategy, traditionally three basic system architectures are widely adopted: deliberative, reactive and hybrid (**?**, **?**). Deliberative systems based on central planning are well suited for the centralized control approach. The single controller makes a plan from its Sense-Plan-

Act (SPA) loop by gathering the sensory information and each robot performs its part. Reactive systems are widely used in distributed control where each robot executes its own controller maintaining a tight coupling between the system's sensors and actuators, usually through a set of well-designed behaviours. Here, various group behaviour emerges from the interactions of individuals that communicate and cooperate when needed. Hybrid systems are usually the mixture of the two above approaches; where each robot can run its own hybrid controller with the help of a plan with necessary information from all other robots. Behaviour-based control architecture can also be considered as a separate category of distributed control architecture (**?**), where each robot behaves according to a behaviour-based controller and can learn, adapt and contribute to improve and optimize the group-level behaviour.
Although most of the MRS control architectures share some common characteristics (such as distributed and behaviour-based control strategy) based on their difference of underlying design principles we have put them into three groups:

1. Behaviour-based classical architectures
2. Market-based architectures
3. Multi-agent based architectures

Due to the overwhelming amount of literature on MRS architectures it is not possible to include all of them. However, below some representative key architectures, strictly designed for MRS, are described.

Behaviour-based classical architectures

The ALLIANCE architecture (**?**) is one of the earliest behaviour-based fully distributed architectures (Fig. **??**). This architecture has used the mathematically modelled behaviour sets and motivational system (Fig. **??**). The primary mechanism for task selection of a robot is to activate the motivational behaviour partly based on the estimates of other robots behaviour. This architecture was designed for heterogeneous teams of robots performing loosely coupled tasks with fault-tolerance and co-operative control strategy. Broadcast of local eligibility (BLE) (**?**) is another behaviour-based architecture that uses port-attributed behaviour technique through broadcast communication method. It was demonstrated to perform

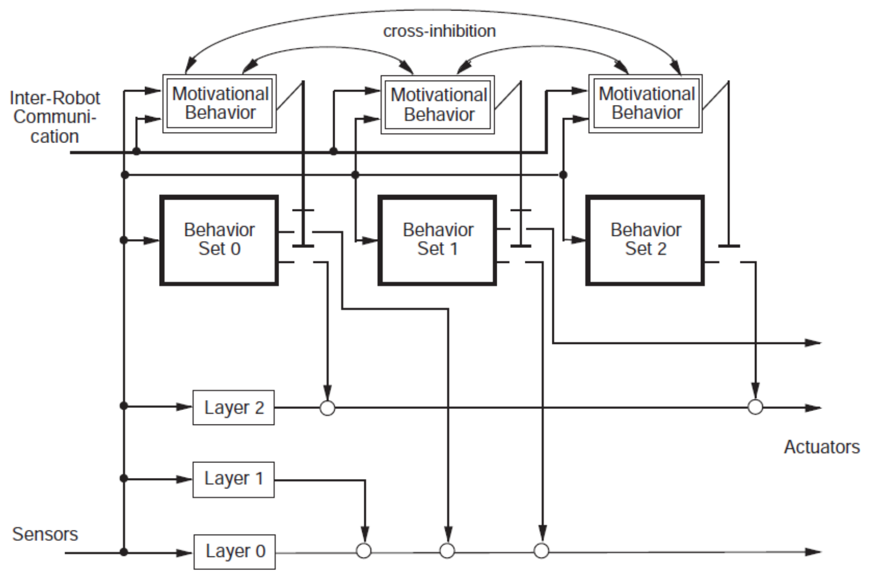


Figure 2.31: ALLIANCE architecture. From **?**.

coordinated tasks, such as multi-target observation tasks. Major differences between this two behaviour-based systems include the need in ALLIANCE for motivational behaviours to store information about other individual robots, the lack of uniform inter-behaviour communication, and ALLIANCE's monitoring of time other robots have spent performing behaviours rather than BLE's local eligibility estimates. Similar to the above two architectures, many other researchers proposed and implemented many variants of behaviour-based architectures. Some of them used the classic three layer (plan-sequence-execute) approach, (e.g. **?**, **?**) used a *layered architecture* where each layer interact directly to coordinate actions at multiple levels of abstraction.

Market-based architectures

Using the theory of marker economics and well-known Contract Net Protocol (CNP) (**?**), these architectures solve the task-allocation problem by auction or bidding process. Major architectures following market-based approaches include MURDOCH (**?**), M+ system (**?**), first-piece auction (**?**), dynamic role assignments (**?**) among others.

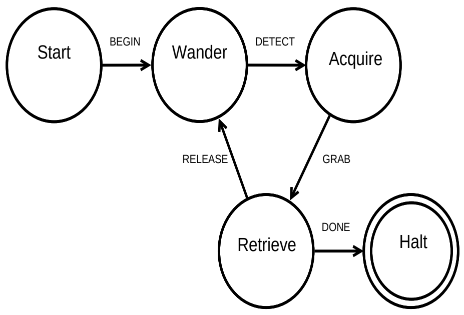


Figure 2.32: Finite state machine for foraging task, reproduced from **?**.

Multi-agent based architectures

Some MRS architectures are influenced by multi-agent systems (MAS). For example, CHARON is a hierarchical behaviour-based architecture that rely on the notion of agents and modes. Similarly CAMPOUT is another distributed behaviour-based architecture that provide high-level functionality by making use of basic low-level behaviours in downward task decomposition of a multi-agent planner. It is comprised of five different architectural mechanisms including, behaviour representation, behaviour composition, behaviour coordination, group coordination and communication behaviours.
In this dissertation, we closely follow the behaviour-based hybrid architecture with an event-driven mechanism for activating behaviours. Our architecture and robot controllers are illustrated in Sec. 3.3.

Interaction and learning

According to the Oxford Dictionary of English the term interaction means reciprocal action or influence. In MRS research, such as in (**?**), interaction is referred to as mutual influence on behaviour. Following this definition, it is obvious that objects in the world do not interact with agents, although they may affect on their behaviour. The presence of an object affects the agent, but the agent does not affects the object since objects, by definition, do not behave, only agents do. However many other researchers acknowledge that interactions of robots with their environment (as found in stigmergic communication) have a great impact on their behaviours. Therefore, we adopt the broad meaning of interaction that is reciprocal action or influence among robots and their environment. From the above review of MRS system architecture, task allocation and communication, it is obvious that interaction among robots and their environment is the core of the dynamics of MRS. Without this interaction, it can not be a functioning MRS. We have split our discussions of MRS interaction into multi-robot learning and communication. Multi-robot learning is described here and communication in MRS is discussed in Sec. 2.5. A great deal of research on multi-robot learning has been carried out since the inception of MRS (**?**, **?**, **?**). Learning, identified as the ability to acquire new knowledge or skills and improve one's performance, is useful in MRS due to the necessity of robots to know about itself, its environment and other team-members (**?**). Learning can improve performance since robot controllers are not perfect by design and robots are required to work in an uncertain environment that all possible states or actions can not be predicted in advance. Besides learning a new skill or piece of knowledge it is also important to forget learned things that are no longer needed or correct as well as, to make room for new things to be learned and stored in a finite memory space of a robot.

Several learning techniques are available in robotics domain, such as reinforce or unsupervised learning, supervised learning and learning by imitation. Although reinforce learning, or learning based on environmental or peer feedback, is a good option for MRS, it has been found that in large teams the ability to lean in this way is restricted due to large continuous state and action space (**?**). Several other learning techniques are also available to explore in MRS domain including Markov models, Q-learning, fuzzy logic, neural nets, game theory, probabilistic or Bayesian theory among others (**?**).

Localization and exploration

Mobile robot systems highly rely on precise localization for performing their autonomous activities in indoor or outdoor. Localization is the determination of exact pose (position and orientation) with respect to some relative or absolute coordinate system. This can be done by using proprioceptive sensors that monitor motion of a robot or exteroceptive sensors that provide information of world representation, such as global positioning system (GPS) or indoor navigation system (INS). Many other methods are also available, such as landmark recognition, cooperative posi-

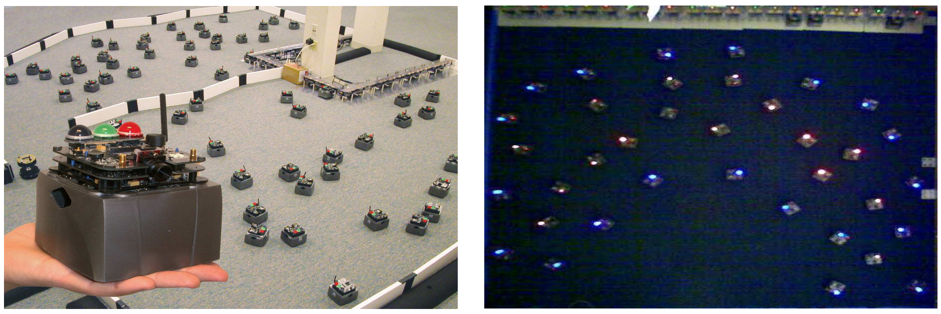

(a) Swarmbot SRS (b) Robots detecting boundary

Figure 2.33: (a) A Swarmbot and the Swarms (b) Swarmbots detecting boundary using distributed algorithms, from **?**.

tioning and other visual methods (**?**).
Localization issue of MRS also invites researchers to examine specific areas like exploration and map generation. In exploration problem, robots need to minimize the time needed to explore the given area. Many researchers uses various kinds of exploration algorithms for solving this NP-hard problem, such as line-of-sight constrained exploration algorithm , collaborative multi-robot exploration (**?**). In mapping problem, mostly inaccurate localization information from teams of robots are accumulated and combined to generate a map by various techniques, such as probabilistic approaches (**?**).

Example areas of MRS research

Researches on MRS have been targeted for numerous application domains that all can not be listed here altogether. Rather than listing all of areas explored by researchers, here we have included a few major areas that have received highest attention in the MRS research community. Cooperative transport of large objects (that one robot is unable to handle) by multi-robots was investigated by many researchers such as, following a formal model of cooperative transport in ants (**?**), box-pushing by six-legged robots (**?**). Another kind of object transport problem include clustering objects into piles e.g., (**?**), collecting waste or trash e.g., (**?**), sorting coloured objects e.g., (**?**), constructing a building site collectively (**?**) and so on. It has also been observed that multi-robot teams as micro or mini machines are helpful to improve the control and efficiency of mining and its processing operations (**?**). Many researchers address MRS research issues under the requirements

of a military or space application. Behaviour-based formation control (**?**), land-mine detection (**?**), multiple planetary rovers for various missions (**?**) and so forth, all are the examples of this areas.
Although research on MRS has been becoming more mature since last decades, it is not easy to find many industrial applications relying on multiple autonomous mobile robots. We have found one exceptional application developed by Kiva Systems in the domain of multi-robot material handling in warehouses (**?**). Along with this, Sec. bg:mrs-industry reviews some possible applications of MRS in automation industry.

2.3.4 Swarm robotic systems (SRS)

Background of SRS: When many traditional MRSes showed serious scalability failures, the necessity of adopting a new paradigm becomes obvious (**?**). Researchers of traditional MRS approach realized that executing their time and processing intensive algorithms in large number of real robots (≥ 10) could be a nightmare. Adding more robots almost exponentially amplified their inherent problems e.g. physical and communication interferences with an ever-ending hunger of more CPU powers. Traditional approach became infeasible for some other reasons too. For example, one of the early inspirations for constructing a MRS is to own 10-20 cheaper and simple robots is preferable to manage 1-2 expensive and complex robots. But under traditional approach, when the robot-team size increases, robots require more sensory and on-board processing power for maintaining large internal and environmental sates. This goes against the original spirit of MRS. Moreover, many traditional MRSes, that relied upon a centralized system for communication, localization and other necessary supports, often failed under heavy-stress conditions of large MRS.
In early and mid-90s, many researchers found that applying biological principles of swarm intelligence effectively removed and reduced many bottlenecks of traditional MRS. In 1995, Maja Mataric published that complex group behaviours could be produced by the appropriate combinations of more simple “basis behaviours” (**?**). The idea of using simple biological behaviours, such as avoidance and following, to create complex flocking and foraging behaviours inspired many other researchers to search solution for controlling large MRS in this direction. The early

research of Reynold (1987) guided many others to apply biological principles of self-organization aka swam intelligence (SI) in MRS. The term *swarm intelligence* was first coined by Gerado Beni (**?**) in late 1980s and it represents the effort of designing algorithms or distributed problem solving devices inspired by the collective behaviours of social insect colonies (**?**). The idea of using simple robots to create complex patterns or structures was also studied under *cellular robotics* (**?**). During recent years the term swarm robotics (SR) emerged as an application of swarm intelligence to multi-robot systems with the emphasis on physical embodiment of entities and realistic interactions among the entities and between the entities and their environment. These systems of swarm robotics or minimalist robotics [1] can be represented by a common term swarm robotic system (SRS).

Advantages of SRS: The simplicity of SRS approach inspires robotic researches to build MRS with cheap robotic hardware, to equip them with simple controllers and control them through local informations without, creating any explicit model of the environment or using any sophisticated central control. The redundancy of robots, parallelism in their task-executions and an overall distributed control architecture, support addition or removal (or failure) of any robots in run-time. Moreover the control algorithms are now decoupled from the model of the environment or other robots. Thus this system now becomes more robust, fault-tolerant, scalable and adaptive to unknown and dynamic environment.

Distinct features of SRS: In order to distinguish swarm robotics from other branches of robotics such as collective robotics, distributed robotics, robot colonies and so forth, **?** proposed a formal definition and a set of criteria for swarm robotics research .

> "Swarm robotics is the study of how large number 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."

And the notable criteria of swarm robotics research are listed as follows.

Autonomous robots that exclude the sensor networks and may include metamor-

[1] although both SR and minimalist robotics follow similar approaches for solving similar problems, the latter does not explicitly relate its origin to SI

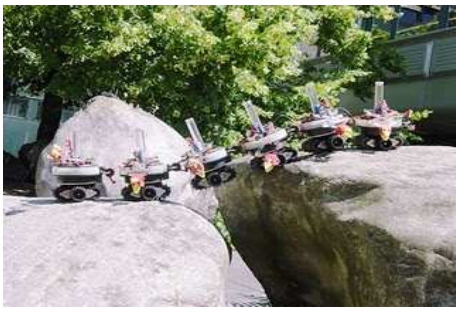


Figure 2.34: A group of Swarmbots are crossing rough terrain, from **?**.

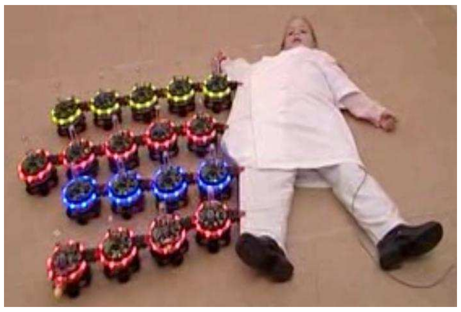


Figure 2.35: A Swarmbot of 18 s-bots pulls a child to a safe location from **?**.

phic robotic system without having no centralized planning and control element.

Large number of robots, usually $\geq$ 10 robots, or at least having provision for scalability if the group size is below this number.

Mostly homogeneous groups of robots that typically exclude the multi-robot soccer teams having heterogeneous robots.

Relatively incapable of inefficient robots that is the task complexity enforces either cooperation among robots or increased performance or robustness without putting no restriction on individual robot's hardware/software complexity.

Robots with local sensing and communication capabilities that does not use global coordination channel to coordinate among themselves, rather enforces distributed coordination.

Classification and application of SRS: SRS can broadly be classified into two distinct classes. The first class consists of simple and relatively inexpensive mobile robots that are fully autonomous and can work in isolation. For example, E-puck robots (**?**) fall under this class. Other class of robots include self-reconfigurable (**?**) and self-assembling robots which can be built by coupling several identical units together, e.g. a robotic snake. In this dissertation, we have limited our focus to the former class of robots alone.
Fig. 2.34 shows amazing abilities of Swarmbot that is crossing a rough terrain. Fig

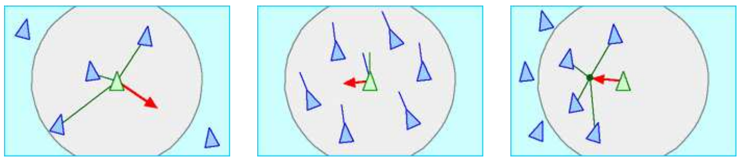

(a) Separation (b) Alignment (c) Cohesion

Figure 2.36: Reynold's simulated flocking of boids **?**. (a) Separation: steer to avoid crowding local flockmates, (b) Alignment: steer towards the average heading of local flockmates and (c) Cohesion: steer to move toward the average position of local flock-mates, from http://www.red3d.com/cwr/boids/, last seen on 01/06/2010.

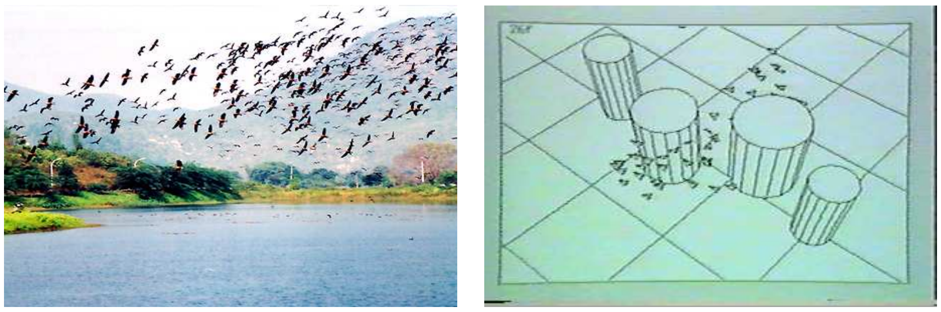

(a) A flock of birds (b) Simulated boids

Figure 2.37: (a) Real flocks of birds, from http://www.travelblog.org, and (b) simulated flock of birds, from http://www.red3d.com/cwr/boids, last seen on 01/06/2010.

2.35 shows another interesting demonstrations of SRS that a team of 18 Swarmbots pulls a child to a safe place.

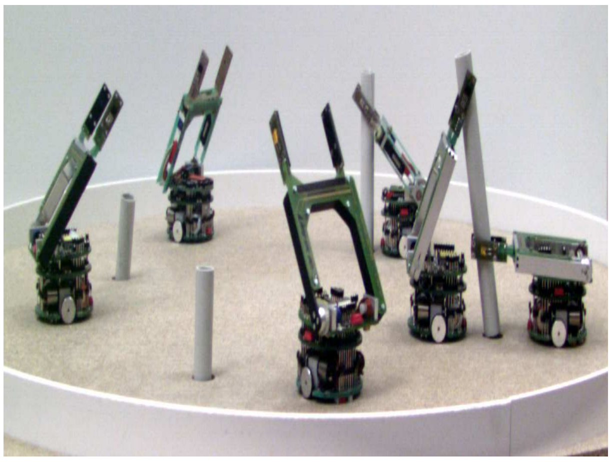
Modelling Swarms: Modelling both the behaviour of an individual robot controllers and system-level collective behaviours have become an interesting issue in SRS. This is due to the fact that, in this kind of system, collective behaviours, e.g. flocking or aggregation, can significantly be changed by of just changing one or few parameters of individual robot controllers. Thus modelling SRS can give us an early insight about a target system before its implementation before doing any time-consuming simulation or expensive real-experiment. This is very important since in real-experiments or in simulations the parameter space usually becomes huge and it could be infeasible to run significant number of iterations of these real or simulated trials. From mathematical models, it can be easy to find out which parameters of interest dominate the system performance or which resources are more important to do a particular type of tasks. In some cases, it can also predict the system performance to some degree.

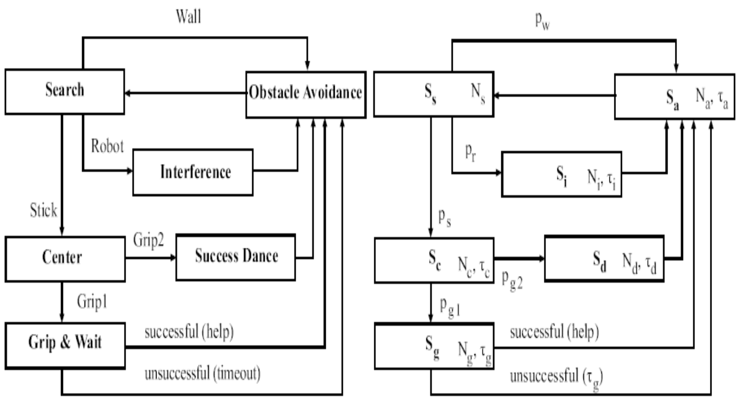
A plenty of approaches for modelling SRS exists in MRS literature (**?**). Most common modelling approaches include: behaviour-based approach, probabilistic models, potential-function based approach, asynchronous swarm models, multi-agent based swarm models. Behaviour-based approach can be found in the study of **?** who has simulated the flocking behaviours of birds. Fig. 2.36 illustrates how three simple rules can produce a coordinated flocking motion (Fig. **??**(b)). From biological observation of flocking birds it is obvious that collective behaviours can be generated through a local control and interaction rules. Similar to this study, many other researchers tries to apply behaviours based local strategy for formation control (e.g. (**?**)), aggregation (e.g. **?**), sorting (e.g. (**?**)), foraging (e.g.Liu+2007), cooperative transport (e.g. (**?**)) etc.

In 1999, **?** proposed probabilistic modelling of SRS. This probabilistic approach often has two major aspects: controller design through probabilistic finite state machine (PFSM) (e.g. see Fig. 2.38) and automated parameter adaptation through genetic algorithm. This approach has been adopted by many recent SRS research (e.g. **?**, **?**, **?**).

SRS models can be classified into many distinct classes. Firstly, they can be classified into: spatial and non-spatial models. *Spatial models* keep track of the agent's trajectories and perhaps use that spatial distribution. However, in *non-spatial models* it is assumed that agents occupy independent, random positions at consecutive time-steps. SRS models can also be classified into embodied and non-embodied models. Here *non-embodied models* consider agents as points and their physical characteristics are ignored, whereas *embodied models* take the physical characteristics or interferences of agents into account. Thus spatial models with embodied agents are chosen in typical simulations.

As another distinct classification, SRS models can be classified into two major groups: 1) microscopic models and 2) macroscopic models. *Macroscopic models* focus on individual robots and state transitions of each robot controller are updated based on the stochastically approximated robot-robot and robot-environment interactions. The probabilities of state transitions are calculated from simple geometric configurations and with few trial experiments. Here, no group-level sensing or actuation is taken into account. On the other hand, *macroscopic models* captures


(a) Robots pulling sticks



(b) Probabilistic FSM

Figure 2.38: (a) Stick-pulling experiments by a group of Khephera robots equipped with gripper turrets and (b) Probabilistic finite state machie (PFSM) of robot controllers. From **?**.

the snapshot-by-snapshot pictures of whole SRS. Each snapshot presents the total number of robots in a given state. Fig. 2.38(b) shows the probabilistic macroscopic model where S_x denotes a particular state x and N_s denotes the number of robots under state S_x. Here τ represents the corresponding probability density function derived from a set of Master-Equations (**?**). Despite a lot of attractive benefits of SRS modelling approaches, they have some notable shortcomings. Formal models of SRS, particularly probabilistic models, may not be attractive or useful for many reasons. Firstly, constructing a functional model takes time due to the need for accurate calibration of necessary parameters (which also involves running several real-experiments or simulations). Secondly, most of probabilistic models rely upon some assumptions, e.g. coverage of the area should be spatially uniform or the system should follow Markov properties i .e. the robot's future state depends only on its current state and how much time it has spent in that state. These can not be satisfied in many practical applications e.g. open space exploration or robots with memory. As task complexity increases the parameters space becomes large and searching good combinations of parameters by some means, e.g. genetic algorithms, becomes more complex.

Similar to traditional MRS, SRS faces great challenges in enabling localization, communication and interaction in group-level. For example, without the presence of any centralized localization module, such as GPS or indoor navigation system (INS), it is not easy to localize precisely and locally the position of a robot with respect to other robots or environment. Recently researchers investigated these issues and reported some novel solutions, e.g. **?** presented a novel technique based on trilateration for localization of swarm robots using ultrasonic and RF transceivers. **?** reported hop-count and bio-inspired strategies for collective perception or how a swarm robot can join multiple instances of individual perception to get a global picture. (**?**) presented a collaborative localization algorithm using landmark based localization technique. Because of the similarity of the problems in both traditional MRS and SRS, we have presented the issues of task-allocation and communication of both types of MRS in Sec. 2.4 and Sec. **??** respectively.

In this dissertation, we closely follow the SRS approach of designing robot group and solving related issues. We have followed the behaviour-based approach for designing robot controllers (Chapter **??**) that rely on GPS-like overhead camera-based solution to fulfil their localization needs. We have modelled our real robotic system considering their spatial, embodied and microscopic properties. No macroscopic simulation or analysis of the robot group has been conducted. Our autonomous robot group meets all the criteria of a SRS mentioned by (**?**) except the communication issue. We do not restrict our robots always to do local communication alone for solving their MRTA problem. Our solutions to MRTA and multi-robot communication problems have been presented in Chapter 4 and Chapter 5.

2.4 Multi-robot task allocation (MRTA)

Since 90s multi-robot task allocation (MRTA) is a common research challenge that tries to define the preferred mapping of robots to tasks in order to optimize some objective functions (**?**). Many control MRS architectures have been solely designed to address this task-allocation issue from different perspectives . Based-on the high-level design of those solutions, here we have classified researches on MRTA into two major categories: 1) predefined or intentional task-allocation and 2) Bio-inspired self-organized task-allocation. Fig. 2.39 illustrates our classification. This classification has been adopted from **?**, but our sub-categories are different since **?** proposed the classification for multi-agent system alone that does not take the spatiality and embodiment of agents into account. Under each of our sub-categories of MRTA there are many inter-connected issues that need to be addressed by the system designer. We have put major issues into three major axes: 1) organization of task-allocation, 2) degree of communication and 3) degree of interaction. In the following subsections, we have discussed these two categories and their key issues with some example MRSes and their comparisons.

2.4.1 Predefined task-allocation

In most of the traditional MRS, task allocation is done using a well-defined models of tasks and environments. Here it is assumed that the system designer has the precise knowledge about tasks, robot-capabilities etc. Many flavours of the type of task-allocation can be found in the literature. Below we briefly discussed a few well-acknowledged works.

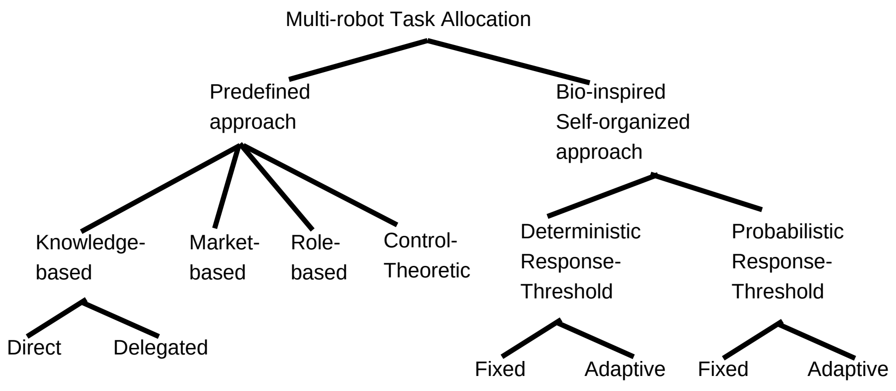

Multi-robot Task Allocation
Predefined approach
Bio-inspired Self-organized approach
Knowledge-based
Market-based
Role-based
Control-Theoretic
Deterministic Response-Threshold
Probabilistic Response-Threshold
Direct
Delegated
Fixed
Adaptive
Fixed
Adaptive

Figure 2.39: Classification of MRTA

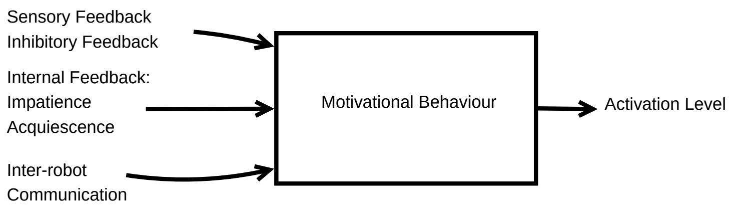


Figure 2.40: Motivational behaviour in ALLIANCE. From **?**.

Knowledge-based and multi-agent based approaches:

In this approach knowledge-based techniques are used to represent tasks, robot capabilities etc. One of the early well-known MRTA architecture of this category was Parker's ALLIANCE (**?**) in which each robot models the ability of team-members to perform tasks by observing their current task performances and collecting relevant task quality statistics e.g. time to complete tasks. Robots use these models to select a task that benefit the group as a whole. As shown in Fig. 2.31, ALLIANCE architecture, implemented in each robot, delineates several mathematically modelled behaviour sets, each of which corresponds to some high-level task-achieving function. The concept of motivational behaviour was introduced as a mechanism to choose among these high level behaviours. As shown in Fig. 2.40, each motivational behaviour had a number of inputs and one output. The output, i.e. the activation level corresponding behavioural set, was activated once a predefined threshold was passed. In the same time, all other behavioural sets became inhibited for allowing that selected behavioural set to complete its task. The input of the behavioural sets was ranged from sensory reading to robot-robot broadcast communication of

state-information. Internal behaviours e.g. *impatience* and *acquiescence* were also used to evaluate the motivation of a robot to select a high-level behaviour set. Impatience encouraged individual robots to perform a task that was not selected by any other robot of the team and a robot's acquiescence of a task was increased when a robot selected to perform it. Moreover, robots had the ability to override the inhibitory signal from another robot if a task assigned to other robot was not being completed to a desired level (e.g. when a robot stalled). In case of unsatisfactory self-progress, i.e. not doing any significant progress in a task, robots were able to switch from that task to a different one. This system was deployed on a mock hazardous waste clean up and achieved fault-tolerant distributed task performance of the robot team. Later on, L-ALLIANCE, an extension of this system was also developed to enable robots to learn from the observations of a set of task-performance metrics.
Similar to ALLIANCE, multi-agent based task allocation also use both centralized and decentralized approaches for allocating tasks among its peers. **?** presented a detailed categorization where in a multi-agent system task allocation can be done by using various agents ranging from a central supervising agent or a few mediator agents to all independent agents. In case of centralized systems, the central supervisor (or a group of mediators) must have the necessary system knowledge, e.g. the capabilities and availabilities of all agents, descriptions of tasks, This system gives a well coordinated,consistent and optimized task-allocation but reduces the reliability and fault-tolerance and scalability of the system. On the other hand, in case of distributed task-allocation, each agent can directly assign a task *directly* to another agent provided that all of them have precise knowledge about others. This approach is very expensive for large number of agents since it requires all agents to have huge processing power and communication bandwidth which is not practical. Alternatively, agents can know only a few agents and *delegate* a task to these known peers so that a suitable agent can be found who has sufficient capabilities and free resources to do this task. This task-allocation by delegation also suffers from poor performance due to the use of time-consuming search algorithms. This approach also assumes the availability of high communication bandwidth which is not true in large systems.

Market-based approaches:

As a feasible alternative to the above common multi-agent based task-allocation techniques, many researchers have been following the market-based bidding approach. **?** provided a survey on it. Originated from Contract-Net Protocol, market-based approach can be implemented as a centralized auctioning system or as a combination of *a few auctioneers – all bidders* or, independently *all auctioneers – all bidders*. For example, in a completely distributed system, when a robot needs to solve a problem or task for which it does not have necessary expertise or resources, it broadcasts a task-announcement message, often with a expiry time of that message. Robots that received the message and can solve that task return a bid message. The initiating robot or *manager* selects one (or more) bidder, called as *contractor*, and offers the opportunity to complete the task. The choice of contractor is done after selection by the manager and mutual agreement that maximizes the individual profits. High-level communication protocol is necessary to define several types of messages with structured content. In centralized market-based approach there is only one manager that can be an external supervising agent or one of the robots. While market-based approach consume more resources it usually produces more efficient task-allocations. Anonymous robots can be selected for tasks and these can be different in each bidding cycle.

Role or value-based task-allocation:
In this type of task-allocation each role assumes several specific tasks and each robot selects roles that best suit their individual skills and capabilities (**?**). In this case, robots are typically heterogeneous, each one having variety of different sensing, computation and effector capabilities. Here robot-robot or robot-environment interactions are designed as a part of the organization. In multi-robot soccer (e.g. (**?**)), positions played by different robots are often defined as roles, e.g. goalkeeper, left/right defender, left/right forwarder etc. The robot, best suited and in closest proximity to available roles/positions, selects to perform that role.

Control-theoretic approaches:
In this type of task-allocation systems, a model of the system is usually developed that converts the task specification into an objective function to be optimized. This model typically uses the rigid body dynamics of the robots assuming the masses

and other parameters well-known. Control laws of individual robots are derived either by analytically or by run-time iterations. Unlike most other approaches where task-allocation problem is taken as discrete, control-theoretic approaches can produce continuous solutions. The formalisms of these systems allow system designer to check the system's controllability, stability and related other properties. These systems typically use some degree of centralization, e.g. choosing a leader robot. Example of control-theoretic approach include: multi-robot formation control (**?**), multi-robot box-pushing (**?**) etc.
Predefined Task allocation through other approaches are also present in robotics literature. For example, inspired by the vacancy chain phenomena in nature, (**?**) proposed a vacancy chain scheduling (VCS) algorithm for a restricted class of MRTA problems in spatially classifiable domains.

2.4.2 Bio-inspired self-organized task-allocation

Task performance in self-organized approaches relies on the collective behaviours resulted from the local interactions of many simple and mostly homogeneous (or interchangeable) agents. Robots choose their tasks independently and asynchronously using the principles of self-organization, e.g. positive and negative feedback mechanisms, randomness (see Sec. **??** for details). Moreover interaction among individuals and their environment are modulated by the stigmergic, local and broadcast communications (more in Sec. 2.5). Among many variants of self-organized task-allocation, most common type is threshold-based task-allocation (**?**). Here, a robot's decision to select a particular task depends largely on its perception of stimulus or demand for a task and its corresponding response threshold for that task. Below, we describe most common forms of threshold-based task-allocation: deterministic response-threshold and probabilistic response-threshold techniques. Both of them can use the fixed values of response-thresholds or they can adapt their response-thresholds over time based on a suitable learning or adaptation mechanisms.

$$\sigma(r, e) = \frac{1}{d(r, e)} \tag{2.3}$$

$$\theta_e = \frac{1}{\mid D_r \mid} \tag{2.4}$$

Deterministic response-threshold:

In this approach, each robot has a fixed or deterministic activation threshold for each task that needs to be performed. It continuously perceives or monitors the stimulus of all tasks that reflect the relative urgencies of tasks. When a particular task-stimuli exceeds a predefined threshold the robot starts working on that task and gradually decreases it stimuli. When the task-stimuli falls below the fixed threshold it becomes inactive for that task. This type of approach has been effectively applied in foraging (e.g. **?**, **?**), aggregation (e.g. **?**). This fixed response-threshold can initially be same for all robots (e.g. in (**?**)), or they can be different according robot capabilities or configuration of the system (e.g. **?**).

From a simple example of this approach in event-handling domain (**?**), we can see how task-stimulus can be encoded in mathematical equations. For example, Eq. 2.3 encodes the stimuli of robot r for task-urgency perception event e, ($\sigma(r, e)$) as inversely proportional to the distance between the robot and the event occurring place. Eq. 2.4 gives the threshold value θ_e (based on a predefined distance value D_r) under which robot selects this particular task or event.

Unlike maintaining a fixed response-threshold, adaptive response-threshold model changes or adapts the threshold over time. Response-threshold decreases often due to performance of a task and this enables a robot to select that particular task more frequently or in other words it learns about that task. Examples of this type of task-allocation can be found in (e.g. **?**, **?**).

$$p_e = \frac{\sigma(r, e)^n}{\sigma(r, e)^n + \theta_e^n} \tag{2.5}$$

Probabilistic Response-Threshold:

Unlike deterministic approach, where robots always respond to a task-stimuli that has a largest stimulus above the threshold, probabilistic approach offers a selection process based-on a probability distribution. For example, in probabilistic response, robots can respond to an event e with probability p_e as in Eq. 2.5 where θ_e is the threshold and n is the non-linearity of the response. Thus, robots always have small nonzero probabilities for all tasks.

In this dissertation we have closely followed this approach with an on-line adaptation mechanism which has been outlined in Chapter 4.

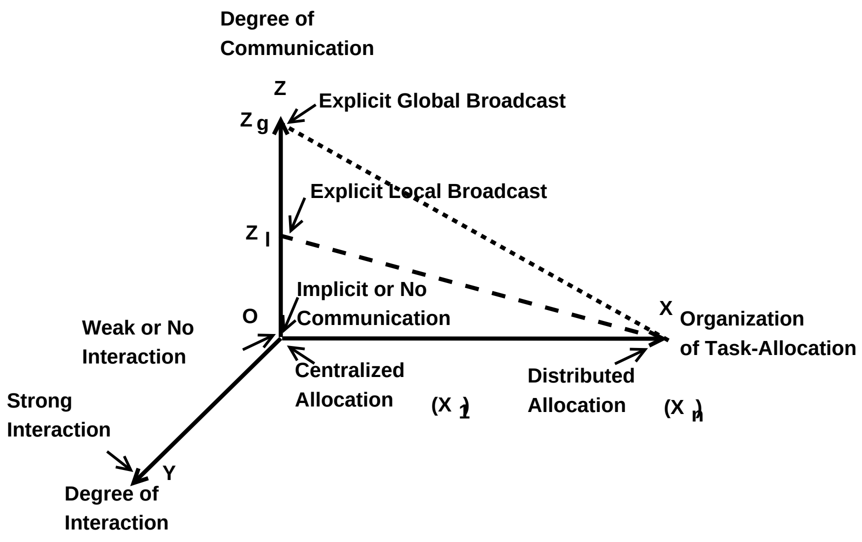


Figure 2.41: Three major axes of complexities in MRTA

2.4.3 Key issues in MRTA

From the vast amount of literature on MRTA, we can easily infer the level of complexities exist in MRTA. In fact researchers generally agree that the MRTA is a *NP-hard* problem where optimal solutions can not be found quickly for large problems (e.g. See **?**, **?**). But why do we find so many variants of MRTA solutions ? In order to answer this question, first we need to look into the contexts from where the solutions are made. Most predefined task-allocation solutions are proposed within the context of a known or controlled environment where the modelling of tasks, robots, environments etc. becomes feasible. Note that here tasks can be arbitrarily complex that often require relatively higher sensory and processing abilities of robots. Robot-team can be consists of homogeneous or heterogeneous individuals, having different capabilities based on the variations in their hardware, software etc. But the uncertainty of the environment is assumed to be minimum. On the other hand, bio-inspired self-organized MRTA solutions are free from extensive modelling of environment, tasks or robot capabilities. Most of the existing research considers very simple form of one global task e.g. foraging, area cleaning, box-pushing etc. This is due to the fact that major focus of this approach is limited mainly to design individual robot controllers in such a way that a few simple or *specific* tasks can be accomplished. More research is needed to verify the capabilities of self-organized approach in doing multiple complex tasks. At this moment, the bottom line remains as “select simple robots for simple tasks (self-organized approach) and complex robots for complex tasks (predefined approach)”.

Both of the above task-allocation approaches expose their relative strengths and weaknesses when they are put under real-time experiments with variable number or robots and dynamic tasks. In an arbitrary event handling domain, **?** compared between self-organized and predefined market-based task-allocation, where they found that predefined task-allocation was more efficient when the information was accurate, but threshold-based approach offered similar quality of allocation at a fraction of cost under noisy environment. **?** presented a comparative study of the complexity and optimality of key architectures, e.g. ALLIANCE (**?**), BLE (**?**), M+ (**?**), MURDOCH (**?**), First piece auctions (**?**) and Dynamic role assignment (**?**), all of them relied upon predefined task-allocation methods. The computational and communication requirements were expressed in terms of number of robots and tasks. Although this study does not explicitly measures the scalability of those key architectures, it clearly shows us that many predefined task-allocation solutions will fail to scale well in challenging environments when the number of robots (and tasks) will increase, under the given limited overall communication bandwidth and processing power of individual robots. In this regard, self-organized task-allocation methods are advantageous as they can provide fully distributed, scalable and robust MRTA solutions through redundancy and parallelism in task-executions. Moreover, the interaction and communication requirements of robots can also be kept under a minimum limit. Thus we can say that for large MRS, self-organized task-allocation methods can potentially be selected, if a system designer can divide his complex tasks into simple pieces that can be carried out by multiple simple robots in parallel with limited communication and interaction needs.

In order to characterize both predefined and self-organized approaches in terms of their deployment, we propose three distinct axes: 1) organization of task-allocation (X), 2) degree of interaction (Y) and 2) degree of communication (Z). Fig. 2.41 depicts these axes with a reference point O. These axes can be used to measure the complexities involved in various kinds of MRTA problems and the design of their solutions. In this figure, X axis represents the number of active nodes that provides the task-allocation to the group. For example, in any predefined task-allocation approach, we can use one external centralized entity or an one of the robots (aka leader) to manage the task-allocation. This can optimize the MRTA solution globally, but is subject to single point of failure. This is also not feasible for large systems where the number of robots and the descriptions of tasks are large. In many predefined methods, e.g. in market-based systems, multiple nodes can act as mediators or task-allocators that we have discussed before. Under predefined task-allocation approach and for a small number of robots, fully distributed task-allocation can also be possible where all nodes can act as independent task-allocator, e.g. as found in ALLIANCE architecture. Most of the self-organized task-allocation methods are fully distributed, i.e. they allocate their tasks independently without the help of a centralized entity. However, they might be dependent on external entities for getting status or descriptions of tasks. Recent studies on swarm-robotic flocking by (**?**) shows that a swarm can be guided to a target by a few informed individuals (or leaders) while maintaining the self-organizing principles for task-allocation. Task-allocation of a swarm of robots just by one central entity may be rare since one of the major spirits of swarm robotic system is to become fully distributed.

The Y axis of Fig. 2.41, corresponds to the level of robot-robot interaction present in the system. As we have mentioned before in Sec. 2.3.2, interaction can be classified into various levels: collective, cooperative, coordinative and collaborative. The presence of interaction can be due to the nature of the problem, e.g. a cooperation is necessary in co-operative transport tasks. Alternately, this interaction can be a design choice where interaction can improves the performance of the team, e.g. cooperation in cleaning a work-site is not necessary but it can help to improve the efficiency of this task. Y axis can also be used to refer to the degree of coupling present in the system (**?**). In case of collective interaction, robots merely co-exist, i.e. they may not be aware of each other except treating others as obstacles. Many other MRS are loosely-coupled where robots can indirectly infer some states of the environment from their team-mates' actions. But in many cases, e.g. in co-operative transport, robots not only recognize others as their team-mates, but also they coordinate their actions. Thus they form a tightly coupled system. This level of interaction and coupling also gives us the information about potential side-effects of failure of an individual robot. Tightly coupled systems where high degrees of interactions among the robots are present suffer from performance loss if some of the robots removed from the system.

The Z axis of Fig. 2.41 represents the communication overhead of the system. This can be the result of the interactions of robots under a given task-allocation method. As we have discussed before various task-allocation methods rely upon variable degrees of robot-robot communications. On the other hand, the communication capabilities of individual robots can limit (or expand) the level of interaction can be made in the given group. Thus in one way, considering the interaction requirements of a MRTA problem, the system designer can select suitable communication strategies that both minimizes the communication overhead and maximizes performance of the group. And in other way, the communication capabilities of robots can guide a system designer to design interaction rules of robot teams, e.g. the specification of robot's on-board camera can determine the degree of possible visual interactions among robots. The suitable trade-offs between these two axes: communication and interaction can give us a balanced design of our MRTA method.

Finding suitable communication strategies under the adaptive response-threshold task-allocation method is the central issue of this thesis. So we have focused to examine the benefits of traversing along the various axes of Fig. 2.41. In this dissertation, we are interested on two distinct lines: 1) distributed task-allocation, with no direct robot-robot interaction and communication, say line OX_n (n being the number of robots) and 2) distributed task-allocation, with no direct robot-robot interaction, but varying degrees of local communications, say line X_nZ_l (Z_l being a local broadcast communication strategy that involves l number of peers in communication). Our MRTA experiments along OX_n and X_nZ_l can be found in Chapter 4 and Chapter 5. The issue of multi-robot communication is presented in more detail in next section.

2.5 Communication in MRS

Communication plays an important role for any high-level interaction (e.g. cooperative, coordinative) among a multi-robot team (**?**). This is not a prerequisite for the group to be functioning, but often useful component of MRS (**?**). The characteristics of communication in MRS can be presented in terms of these issues:

- Rationale of communication: (*why do the robots communicate*),
- Message content (*what do they communicate*),
- Communication modalities (how do they communicate), and
- Target recipients (*With whom do they communicate*).

Below we have described the above issues with a focus on how communication can lead to effective multi-robot task allocation.

2.5.1 Rationale of communication

From three kinds of communication experiments: indirect stigmergic communication, direct robot to robot state communication, and goal communication, (**?**) found that in some tasks communication provided performance improvements while others did not. Most of the robotic researchers generally agree that communication in MRS usually provides several major benefits, such as:

Improved perception: Robots can exchange potential information (as discussed below) based on their spatial position and knowledge of past events. This, in turn, leads to improve perception over a distributed region without directly sensing it.

Synchronization of actions: In order to perform (or stop performing) certain tasks simultaneously or in a particular order, robots need to communicate, or signal, to each other.

Enabling interactions and negotiations: Communication is not strictly necessary for collective interactions of a robot team. From a set of multi-robot communication experiments **?** concluded that for certain classes of tasks, explicit communication is not strictly necessary. However, higher level interaction, such as cooperative task-performance or coordinative actions are almost always designed with communication support built-into the robots (see Sec. 2.3.2 for definition of various kinds of interactions). Thus communication can help a lot to influence each-other in a team that, in turn, enables robots to interact and negotiate their actions effectively.

Figure 2.42: A team of s-bots communicating by light signals. From http://lis.epfl.ch, last seen on 01/06/2010.

Figure 2.43: A fleet of robots relying on camera (vision) for search operation. From http://www.cs.utk.edu, last seen on 01/06/2010.

2.5.2 Information content

Although communication provides several benefits for team-work it is costly to provide communication support in terms of hardware, firmware as well as run-time energy spent in communication. So robotic researchers carefully minimize the necessary information content in communications by using suitable communication protocols and high-level abstractions. For example in foraging, grazing and consuming experiments **?** introduced state and goal communications. In state communication, a single bit is transmitted indicating the current state of robot (e.g. 0 transmitted when robot was in *Wander* state and 1 transmitted when robot was in *Acquire* or *Retrieve* states). In case of goal communication, the localization of task was also transmitted. From similar instances of these experiments below is a brief summary of information contents.

- **Individual state:** ID number, battery level, task-performance statistics, e.g. number of tasks done.
- **Goal:** Location of target task or all tasks discovered.
- **Task-related state:** The amount of task completed, number of other robots present there etc.
- **Environmental state:** Free and blocked paths, level of interference found, any urgent event or dangerous changes found in the environment.

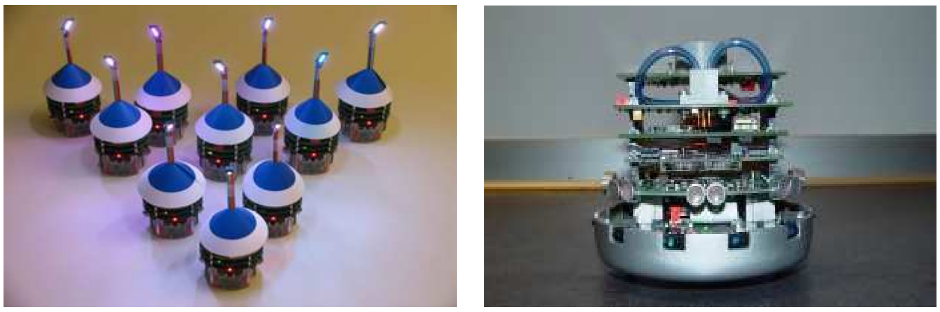

(a) E-puck robots with table-lamps (b) Sniffing Khepera III

Figure 2.44: (a) A fleet of mobile “lighting” robots moving on a large table, such that the swarm of robots form a distributed table light and (b) Distributed odour source localization by Khephera robot equipped with volatile organic compound sensor and an anemometer (wind sensor). From http://http://disal.epfl.ch, last seen 01/06/2010.

- **Intentions:** Detail plan for doing a task, sequences of selected actions etc.

Since a MRS can be comprised of robots of various computation and communication capabilities, it communication content can vary greatly based on their individual communication modules and channel capacities.

2.5.3 Communication modalities

Robotic researchers typically use robot’s on-board wireless radio, infra-red (IR), vision and sound hardware modules for robot-robot and robot-host communication. The reduction in price of wireless radio hardware chips e.g. wifi (ad-hoc WLAN 802.11 network) or Bluetooth [2]makes it possible to use wireless radio communication widely. In-expensive IR communication module is also typically built into almost all mobile robots due to its low-cost and suitability for ambient light and obstacle detection. IR can also be used for low bandwidth communication in short-range, e.g. keen-recognition. Most robots can also produce basic sound waves and detect it with their built-in speakers and suitable configuration of on-board microphones. Although speech-recognition is not commonly found in mobile robots yet, detection of pre-recorded sound waves can be feasible.

Most of the mobile robots come with a series of LEDs, and tiny camera that can emit light signals and detect it with camera. Fig. 2.42 shows the robot-robot com-

[2]www.bluetooth.com

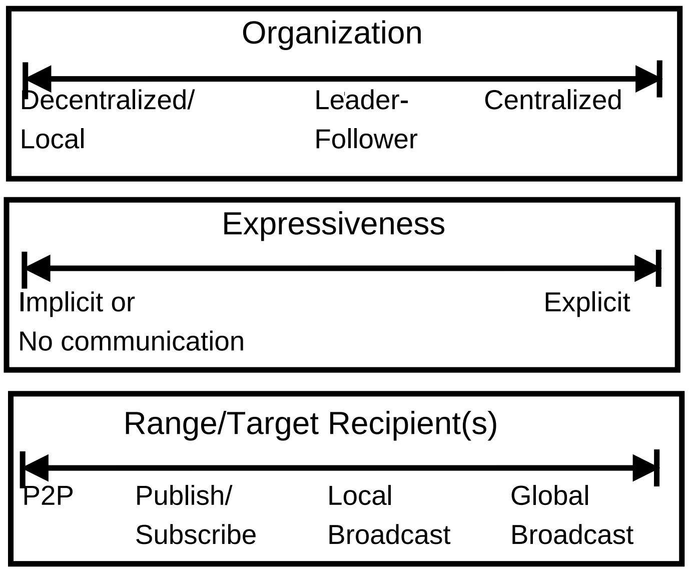


Figure 2.45: Three aspects of communication in MRS

munication through the red and green coloured LEDs. Many robots can also detect blobs of colours and can recognize peers of other objects through the use of a color-coded markers. Fig. 2.43 shows a team of robots with colour-coded markers attached on it that can be detected by the on-board camera of other robots. Some other researchers also tried to establish communication among robots solely relying on vision (**?**). Although a lot of researches have been carried out to design robot skin and tactile communication system, we do not know any instance of tactile communication system in MRS. However, **?** showed limited success in odour-source localization, a form of detecting chemical signals. Apart from that, we do not know about any full-fledged chemical communication in any MRS.

2.5.4 Communication strategies

Whatever be the communication need or modalities in a MRS, suitable strategies are required to disseminate information in a timely manner to a target audience that maximizes the effective task-completion and minimizes delays and conflicts. A review of various communication strategies in social system has been presented in Sec. 2.1.2. Here, in order to discuss the complexities of communication strategies we have selected three independent scales: organization, expressiveness and range of communication. With these independent aspects, we can measure the level of complexities in communication and classify a MRS according to its communication characteristics. Fig. 4.3 outlines these scales and we have described them below.

Centralized and decentralized communications

Similar to the organization of MRTA, communication in a MRS can be organized using an external/internal central entity (e.g. a server PC, or a leader robot) or, a few leader robots, or by using decentralized or local schemes where every robot has the option to communicate with every other robot of the team. From a recent study of multi-robot flocking **?** have shown that a mobile robot flock can be steered toward a desired direction through externally guiding some of its members, i.e. the flock relies on multiple leaders or information repositories. Note that here task-allocation is fully decentralized i.e. each robot selects its task, but the communication structure is hybrid; robots communicate with each other and a centralized human interaction is also present.

Explicit and implicit communications

Communication in a MRS can also be characterized its expressiveness or the degree of explicitness. In one extreme it can be fully implicit, e.g. stigmergic, or on the other end, it can be fully explicit where communication is done by a rich vocabulary of symbols and meanings. Researchers generally tend to stay in either end based on the robotic architecture and task-allocation mechanism used. However, both of these approaches can be tied together under any specific application.

Explicit or direct communication: This is also known as intentional communication. This is done purposefully by usually using suitable modality e.g. wireless radio, sound, LEDs. Because explicit communication is costly in terms of both hardware and software, robotic researchers always put extra attention to design such a system by analysing strict requirements such as communication necessity, range, content, reliability of communication channel (loss of message) etc.

Implicit or indirect communication: This is also known as indirect stigmergic communication. This is a powerful way of communication where individuals leave information in the environment. This method was adopted from the social insect behaviour, such as stigmergy of ants (leaving of small amount of pheromone or chemicals behind while moving in a trail).

Local and Broadcast communications

The target recipient selection or determining the communication range (or sometimes called radius of communication) is an interesting issue in MRS research. Researchers generally tries to maximize the information gain by using larger range. However, transmission power and communication interference among robots play a major role to limit this range. The following major instances of this strategy can be used.

- **Global broadcast communication:** where all robots in the team can receive the message.
- **Local broadcast communication:** where a few robots in local neighbourhood can receive the message.
- **Publish-subscribe communication:** where only the previously subscribed a few robots can receive the message.
- **Peer-to-peer communication:** where only the closest peer robot can receive the message.

2.5.5 Key issues in MRS communication

In multi-robot communication researchers have identified several issues. Some of the major issues are discussed here.

Determination of local neighbourhood

Most swarm-robotic researchers, who use algorithms based on local-neighbourhood of communication, face this problem of defining the range of local neighbourhood. **?** presented that larger communication range is not always optimum for some types of tasks e.g. exploration where a large number of recipient robots decreased the performance of exploration task. **?** provided a design of optimal communication range of homogeneous robots based on their spatial and temporal analyses of information diffusion within the context of cooperative tasks in a manufacturing shop-floor. Spatial design tried to minimize the time for information transmission and temporal design tried to minimize the information announcing time to avoid excessive information diffusion. Eq. **??** describes their optimal range $\chi_{optimal}$ as a function of information acquisition capacity of robots (c) and the probability of information output of a robot (p). Here c is an integer representing the upper-limit of number of robots that can be the target recipents at any time without the loss of information and $\chi_{optimal}$ gives the average number of robots within the output range.

$$\chi_{optimal} = \frac{\sqrt[c]{c!}}{p} \tag{2.6}$$

Kin recognition

Kin recognition refers to the ability of a robot to recognize immediate family members by implicit or explicit communication or sensing (**?**). In case of MRS, this can be as simple as identifying other robots from objects and environment or as finding team-mates in a robotic soccer. This is an useful ability that helps interaction, such as cooperation among team members.

Representation of languages

In case of effective communication several researchers also focused on representation of languages and grounding of these languages in physical world. Implicit communication generally has no or very little necessity of symbol grounding. In foraging experiments **?** used certain cues or events to adapt the response-thresholds of robots. For example, successfully retrieving food makes a robot keep foraging and colliding with other robots makes a robot more likely to rest. These simple cues are based on dynamic events but they can hardly be adopted in complex tasks and environments. Explicit communication often uses custom hand-crafted messages by a set of symbols. Based on a shared vocabulary, they can provide the necessary meaning for the robots. For new kinds of messages one always needs to modify the symbols and shared vocabulary. In such cases knowledge representation techniques and tools can be useful to some extent (**?**).

Fault-tolerance and reliability

Since every communication channel is not free from noise and corruption of messages significant attention has been also given to manage these no communication situations, such as by setting up and maintaining communication network, managing reliability and adaptation rules when there is no communication link available. In terms of guaranteeing communication, researchers also tried to find ways for a deadlock free communication methods (**?**), such as signboard communication method (**?**).

2.5.6 Role of communication in MRTA

Although robotic researchers have been adopting various communication strategies for achieving MRTA in different task domains, very few studies correlate the role of communication with the effectiveness of MRTA. This is due to the fact that researchers usually adopt a certain task-allocation method and they limit their use of communication strategy to either explicit global/local broadcast (in most predefined task-allocation researches) or implicit/no communication (in most self-organized task-allocation researches). In the former one, communication becomes the part and parcel of the robot-robot interactions that enable them to exchange variety of information as discussed before. But in the latter one, the environment serves as a shared memory for all robots to access information or sense the current state of the environment, mostly locally. Here we have attempted to scrutinize how MRTA has been affected by the variations in communication strategies.

As mentioned in Sec. 2.4.2, **?** empirically studied the comparative performance of MRTA under both predefined and self-organized approaches with event-driven simulations. They found that the accuracy of information is crucial for predefined market-based approach where every robot communicated with every other robot (i.e. global broadcast). In case of unreliable link or absence of communication, threshold-based approach performed same as market-based approach, but with less computational overhead. This indicates the dependence of predefined approach on a reliable communication link. However their global broadcast strategy is not feasible for large teams of real-robots. In case of varying robot's communication range, they found that market-based approach performed well for a short communication-range where robots were able to communicate with less than a third of the total number of team-mates. Since the events are handled based-on their spatial locations only, it is not clear how this strategy will perform in other task-domains. Instead of considering only spatial distance in the task-allocation process, it would be more realistic if an explicit form of urgency were assigned on events based on the associated urgency of those tasks.

In order to pursue MRTA, robots can receive information from a centralised source (e.g. **?**) or from their local peers (e.g. **?**). This centralized communication system is easy to implement. It simplifies the overall design of a robot controller. However as we mentioned before, this system has disadvantage of a single point of failure and it is not scalable. The increased number of robots and tasks cause inevitable increase in communication load and transmission delay. Consequently, the overall system performance degrades. On the other hand, uncontrolled reception of information from decentralized or local sources is also not free from drawbacks. If a robot exchanges signals with all other robots, it might get the global view of the system quickly and can select an optimal or near optimal task. This can produce a great improvement in overall performance of some types of tasks e.g., in area coverage (**?**). But this is also not practical and scalable for a typically large MRS due to the limited communication and computational capabilities of robots and limited available communication bandwidth of this type of system.

A potential alternate solution to this problem can be obtained by decreasing the number of message recipients on the basis of a local communication range. This means that robots are allowed to communicate only with those peers who are physically located within a pre-set distance. When this strategy is used for sharing task information among peers, MRTA can be more robust to the dynamic changes in the environment and energy- efficient (**?**). Simailar to this, **?** reported a distributed multi-robot learning scenario with two cases: 1) robots were allowed to communicate with any two other robots (*Model A*) and 2) robots were allowed to communicate with all robots in a fixed radius (*Model B*). In simulation and real robotic experiments with 10 robots and communication ranges of 0.3 m, 1.0 m and 3.3 m, they showed that Model B performed better in intermediate communication range. However, these learning process of individual robot controller were conducted in static environment and it was not clear why intermediate communication range performed better than other ranges.

Many robotic researchers tried to use some form of adaptation rules in local communication to avoid saturation of the communication channel, e.g. based on robot densities in a given area. As mentioned before, **?** tried to formalize the suitable communication range based on spatial and temporal properties of information diffusion of a given communication channel. The major focus of this type of research is to measure the cost of communication based on some metrics, e.g. transmission time and collisions with other robots, and then regulate communication strategies or ranges dynamically ranging from global broadcast to local P2P or (no communication at all when a huge cost is involved). These ideas are attractive to maximize information gain in dynamic environment, but there is no point of doing communication if there is little or no task-requirement. Thus we find this approach, i.e. maximizing information gain, is not always useful or necessary for effective MRTA.

(**?**) also acknowledged the above fact within the context of their ant-based clustering experiments. They used two simple communication strategies: 1) simple memory sharing by robots (shared memory access) and 2) shared used of environment maps (global sensing). In both of these cases, it was found that communication is only useful when some initial random clustering phase was passed. The accuracy of shared information in highly dynamic environment was poor and did not carry any significant advantage. In case of local memory sharing by robots, they showed that sharing information within a limited number of robots produced more efficient clusters, rather than not sharing information at all in stigmergic communication mode. However, here the urgency of clustering tasks or amount of incomplete task did not influence information sharing among robot and thus when clustering process was about to be finished or completely finished, the communication load on the system remained same.

2.6 Application of MRS in automation industry

Automation industry, particularly distributed automated manufacturing domain, provides an excellent area where MRTA problem can be studied and applied. Most of the research in this area is inspired by intelligent multi-agent technology (**?**). A few other researchers also tried to apply the concepts of biological self-organization

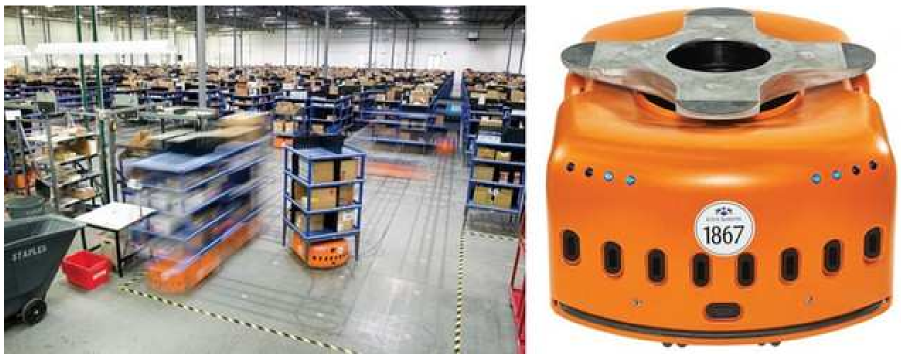


Figure 2.46: A multi-robot material handling system from Kiva systems Inc. From http://www.kivasystems.com/, last seen on 06/06/2010.

(**?**, **?**). In this section we have reviewed these concepts and technologies mainly focusing on physical embodiment of agents, i.e., the use of multiple mobile robots or automated guided vehicles (AGV). The use of static robots (or manipulators) or scheduling of static processes or resources are excluded from this review.

2.6.1 Multi-agent based approaches

Since early 80s researchers have been applying agent technology to manufacturing enterprise integration, manufacturing process planning, scheduling and shop floor control, material handing and so on(**?**). An agent as a software system that communicates and cooperates with other software systems to solve a complex problem that is beyond the capability of each individual software system (**?**). Most notable capabilities of agents are autonomous, adaptive, cooperative and proactive. There exists many different extensions of agent-based technologies such as holonic manufacturing system (HMS) (**?**). Here, a *holon* refers to as an autonomous and cooperative unit of manufacturing system for transporting, transforming, sorting and/or validating information and physical objects.

Agent based technologies have addressed many of the problems encountered by the traditional centralized method. It can respond to the dynamic changes and disturbances through local decision making. The autonomy of individual resource agents and loosely coupled network architecture provide better fault-tolerance. The inter agent distributed communication and negotiation also eliminate the problem of having a single point of failure of a centralized system. These facilitate a manufacturing enterprise to reduce their response time to market demands in globally competitive market. Despite having so many advantages, agent-based systems are still not widely implemented in the manufacturing industry comparing to the other similar technologies, such as distributed objects and web-based technologies due to the lack of integration of this systems with other existing systems particularly real-time data collection system, e.g., radio frequency identification (RFID), supervisory control and data acquisition (SCADA) (**?**). Another barrier is the increased cost of investment in exchange of some additional flexibility and throughput (**?**).

A case study: Kiva material handling system.

As a notable exception in multi-robot automation application, the Kiva material handling system [3] has revolutionized the traditional warehouse order-processing jobs by replacing the old-style relatively expensive AGVs with cheaper mobile robots (**?**). As shown in Fig. 2.46 hundreds of mobile robots are working in a warehouse where thousands of customers orders are handled in real-time. The major advantages of this system over traditional material handling systems are as follows.

- Multiple orders are handled in parallel with random access to all items. This greatly simplifies the ware-house operations.
- Each worker does his job independently. No inter-dependency among workers which is common in sequential order processing system.
- Instead of having 5-10 expensive AGVs, now any company can have 30-50 mobile robots that greatly increases the productivity of the ware-house while reducing operational cost significantly.
- Real-time order processing removes any need for batch processing.
- All major benefits of a distributed system including: no single point of failure, spatial flexibility, expandability.

Kiva material handling system has been implemented using Java-based multi-agent and AI techniques. It uses a centralized resource and task-allocation unit which assigns jobs to mobile robot's drive unit agent and resources to worker's inventory

[3]http://www.kivasystems.com/

station agents using utility-based heuristics, similar to the predefined task allocation approach. Here utility is measured in terms of the cost to the warehouse owner. Spatial location of robots and inventory stations, specific task skills of certain stations, commonalities among orders etc. are taken into account to find the lowest utility cost and to make allocation decisions on the fly. The control architecture of robotic drive units follows typical three layer hybrid control strategy (**?**). In order to locate resources and collectively [4] process customer orders, robotic agents communicate with each other via XML messages. About 100 types of messages have been designed to meet this communication need.
At the time of writing this dissertation, the base price of this system with 30-50 robotic drive units has been estimated as $1M US dollars and this system has been installed about 10-20 warehouses with more than a thousand robotic units being operational [5]. Interested readers may find more on Kiva material handling system and its technical implementation in **?**.

2.6.2 Biology-inspired approaches

The insightful findings from biological studies on insects and organisms have directly inspired many researchers to solve problems of manufacturing industries in a biological way. These can be categorized into two groups: one that allocates task with explicit potential fields (PF) and another that allocate tasks without specifying any PF. Below we have discussed both types of BMS.

Explicit potential filed based BMS

The biological evidences of the existence of PF between a task and an individual worker such as, a flower and a bee, a food source and an ant, inspired some researchers to conceptualize the assigning of artificial PF between two manufacturing resources. For example, PF is assumed between a machine that produce a material part and a worker robot (or AGV) that manipulates the raw materials and finished products. (**?**) conceptualized this PF as the attractive and repulsive forces based on machine capabilities and product requirements. Task allocation is carried

[4]Recall the collective interaction from Sec. 2.3.2 where agents are unaware of each-other, share common goals and their actions are beneficial to team-mates

[5]http://www.roboticstrends.com/, last accessed on 10/06/2010.

out based on the local matching between machine capabilities and product requirements. Each machine generates an attractive field based on its capabilities and each robot can sense and matches this attractive filed according to the requirements of a product. PF is a function of distance between entities. Here, self-organization of manufacturing resources occurred by the process of matching the machine capabilities and requirements of moving robots. Through computer simulations and a prototype implementation of a line-less car chassis welding (**?**) found that this system was providing higher productivity and cost-effectiveness of manufacturing process where frequent reconfiguration of factory layout was a major requirement. This approach, was also extended and implemented in a supply chain network and in a simulated ant system model where individual agents were rational agents who selected tasks based on their imposed limitations on sensing.

BMS without explicit potential fields

Several other researchers did not express the above PF for task allocation among manufacturing resources explicitly, rather they stressed on task selection of robots based on the task-capability broadcasts from the machines to the worker robots. In case of (**?**), task capabilities are expressed as the required time to finish a task in a specific machine. They used assigned priority levels to accomplish the assembly of different kinds of products in the computer simulation of their bionic manufacturing system. In another earlier computer simulated implementation of swarm robotic material handing of a manufacturing work-cell, (**?**) pointed out several pitfalls of such a BMS system, such as dead-lock in manufacturing in inter-dependant product parts, unpredictability of task completion, energy wastage of robots wandering for tasks etc. Although most of these problems remain unsolved researchers are still exploring the concepts BMS in order to achieve a higher level of robustness, flexibility and operational efficiency in a highly decentralized, flexible, and globally competent next generation automated manufacturing system.
In this dissertation, we have considered a virtual manufacturing shop-floor scenario where robots are required to attend virtual production and machine-maintenance jobs in different machines in an arena using their *homing* behaviours. The simplicity of this scenario keeps our implementation clean by separating the MRTA algorithm from the robot capability specific codes. Our detail implementation of

shop-floor scenario is discussed in Chapter 4.

CHAPTER 3

Experimental Tools & Technologies

3.1 General methodological issues

3.1.1 MRS research roadmap

Most of the researches on MRS are empirical in nature. In Sec. **??** we have discussed various trends and paradigms of MRS research. Here we have discussed briefly a general research roadmap using real-robotic hardware. As shown in Fig. 3.1, this roadmap illustrates the details of MRS research process by answering a few basic questions.

Firstly, what is the target application domain of this MRS ? This precisely formulates the target problem to be tackled within this application domain (Fig. 3.1: step 1). Some MRS researches may try to implement and verify the performance of an algorithm inspired from biological social systems, while others may not. Some MRS may focus to solve real-life problems like emergency search and rescue in a disaster site, while others may concentrate on increasing productivity of a manufacturing shop-floor. The selection of this application domain will most likely determine the tasks to be done by individual or group of robots. This will lead to select suitable MRS paradigm that can provide suitable solutions to the problem, and this selected paradigm will indicate whether we need powerful robots with rich sensors/processors or relatively incapable cheap robots for doing some particular tasks (Fig. 3.1: step 2). These tasks and robot group characteristics can be de-

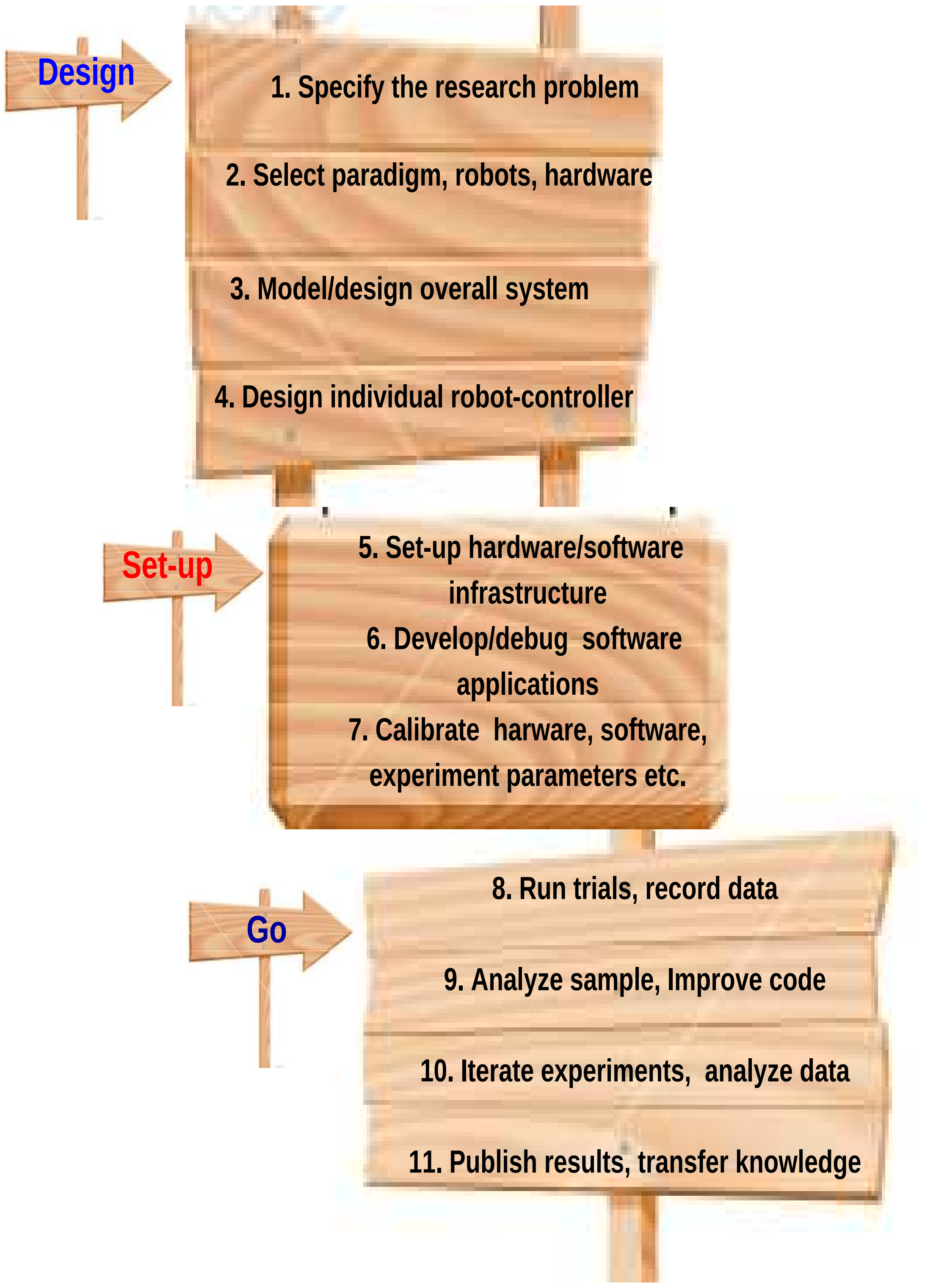

Design
1. Specify the research problem
2. Select paradigm, robots, hardware
3. Model/design overall system
4. Design individual robot-controller
Set-up
5. Set-up hardware/software infrastructure
6. Develop/debug software applications
7. Calibrate harware, software, experiment parameters etc.
Go
8. Run trials, record data
9. Analyze sample, Improve code
10. Iterate experiments, analyze data
11. Publish results, transfer knowledge

Figure 3.1: Stages of MRS research using real-robotic hardware

scribed by using any existing MRS taxonomies, e.g., taxonomies provided by (**?**) and (**?**) (see Sec. 2.3.2) .

Secondly, what are the working models or overall organizing principles (i.e., control architecture) of the robot group and individual robots ? From the task requirements and selected robot's capabilities, one need to fit the MRS into a suitable models of control and architecture (Fig. 3.1: step 3). In Sec. **??** we reviewed most common modelling techniques of MRS, i.e. behaviour-based or probabilistic modelling approaches with three kinds of architectures of control: classic knowledge-based, traditional market-based/role-based and bio-inspired swarm robotic paradigm. Upon selecting a model, most important characteristics of target MRS: e.g., design of individual robot controller, robot-robot and robot-environment communication and coordination patterns etc. will be revealed . This can be achieved by following a top-down or bottom-up design methodology (**?**). For example, one can identify the group behaviours first and then devise individual robot controllers from it. Alternately, robot controllers can be designed first then some formal methodologies can be applied to find the group behaviours (Fig. 3.1: step 4).

Thirdly, what enabling tools and technologies (hardware and software) are required to function the whole robot group autonomously ? Whatever be the architectural design of a MRS, we need to ensure that whole group can maintain the necessary level of task performance through the desired interaction and communication strategies. For reducing cost and other practical reasons, individuals robots may not have the capabilities to localize itself without the help of an external GPS or camera etc. The enabling hardware/software technologies will make-up this individual robot's short-comings (Fig. 3.1: step 5). Moreover, based-on a selected communication technology we need to set-up necessary robot-robot and robot-environment inter-networking infrastructure, e.g., network switches, gateways etc. In order to observe and record experiments and its data for further analysis and improvement, researchers may also employ extra hardware and software logging tools.

Fourthly, what will be the overall software architecture and how the individual components will be implemented ? This will lead to design or select suitable robot control software architecture and programming language(s) for implementing that architecture (Fig. 3.1: step 6). Many existing frameworks are available for low-level robot control, e.g. player/stage [1]. However, high-level robot control often requires custom application development that glues the application code to the low-level robot control code following a solid debugging process.

Fifthly, what hardware/software or control parameters are required to be calibrated ? Often robotic researchers use certain hardware, software that need to calibrated (Fig. 3.1: step 7). For example, to use a camera one needs to calibrate camera and lighting conditions. Low-level calibration of robot sensors may also be required. Suitable software scripts can automate this process and save huge time during real experiments.

Thus, after setting-up a working infrastructure one can think of running real-robotic experiments Initial trials provide sample data that need to verified to find hidden software bugs and run-time errors (Fig. 3.1: step 8). This can be tedious if proper logging and error-checking system is not employed in code. However, development of source-code control tools make it easier to track changes in software code and improve them (Fig. 3.1: step 9). Finally, we can get the potential publishable results after carrying out sufficient iterations of experiments and analysing them properly (Fig. 3.1: step 10-11).

3.1.2 Real robotic experiments vs. simulations

Traditionally robotic researchers use software simulation to validate their model before stepping into real-robotic experiments. Simulating a model by software code is easier and much faster compared to real-world experiments. It does not require any sophisticated hardware set-up or time-consuming software debugging (as shown in Fig. 3.1, step 5-7 can be skipped) . However, in recent times, the abundance of real robot hardware and other necessary tools encourages researchers to test their work in the real systems from the inception of their models. Here we briefly summarize the reasons why we do not follow the traditional "simulation first" approach. Instead of comparing both approaches extensively, we present our rationale behind doing all our experiments in real hardware.

Firstly, contemporary state-of-the-art agent-based simulation packages are essentially discrete-event simulators that execute models serially in a computer's CPU

[1] http://playerstage.sourceforge.net/

(**?**). However in real-world systems agents act in parallel and give the robots: "what you see is what you act upon" environment. In simulations that might not be case. Secondly, the robot-robot and robot-environment interactions are complex and completely unpredictable in real environment than their simulation counterparts. Unexpected failures and inter-agent interactions will not occur in simulations that can either cause positive or negative effects in the experimental results (**?**).
Thirdly, it is not easy to faithfully model communication behaviours of agents in simulations. In our case, we use short-range Bluetooth communication system which is subject to dynamic noise and limited bandwidth conditions.
Fourthly, the dynamic environment conditions, e.g., increased physical interferences of larger team of robots, can also influence the experiment's outcome which may not become obvious in simulations.
Finally, we believe that the algorithm tested in real-robots can give us strong confidence for implementing them in real-robotic systems and later on, this can also be extended or verified in simulations. However, conversely speaking, algorithm implemented in simulation has no warranty that this might work practically with large number of robots.

3.1.3 Design of MRS

Task specification. We intend to design our target MRS for emulating multi-robot manufacturing scenario where robots are required to perform some shop-tasks in different machines while maintaining an effective MRTA. The notion of *tasks* has been kept very simple as our robots are not capable of doing many high-level practical tasks e.g. gripping or recognizing objects, carrying loads etc. Many researchers use additional hardware modules with their robots e.g., gripper to collect pucks or any small objects from floors. Rather than emulating such trivial acts, we have concentrated on the generic and abstract implementation of our algorithm, that can later be fitted to do any real job. Doing real manufacturing tasks has been kept as a future research issue. Thus by "doing a task" our robots usually perform two functions: 1) navigate to a fixed task-location in the experiment arena (hereafter called *navigation*), and 2) they do so by avoiding any dynamic obstacle hereafter called *obstacle avoidance*. In MRS literature this is considered as an equivalent of *homing* behaviour (e.g. **?**). Depending on the time-out value of doing a task, a robot can wait at task-location if it arrives earlier or may switch to a different task and change direction on-the-fly. These symbolic tasks can be mapped to any suitable real task in multi-robot manufacturing domain, such as, material handling or attending a machine for various production or maintenance jobs e.g., welding different machine parts, cleaning or doing maintenance work of a machine etc.

Robot-team specification. Based on the task-requirements we have selected a simple miniature mobile robot, E-puck (**?**), that can do the above navigation tasks avoiding any dynamic or static obstacle. According to the classification of (**?**) our system can be described as below.

- **`SIZE-INF:`** The robot team size is larger than 2 robots and the number of potential tasks. Actually we have used 8 to 16 robots in two sets of experiments and the average number of robots per task has been kept as 4.
- **`COM-INF:`** Robots can communicate with any other robot.
- **`TOP-ADD:`** Every robot can communicate with any other robot by name or address (link path). Naming of robot's communication link path is assumed to follow any simple convention, e.g., /robot1, /robot2 etc.
- **`BAND-INF:`** Every robot can communicate with other robot with as much bandwidth as necessary. Since our robots need to exchange simple messages we ignore this bandwidth issue.
- **`ARR-DYN:`** Robot can change the arrangement dynamically.
- **`CMP-HOM:`** Each robot is initially identical in both hardware and software, but they can become different gradually by learning different tasks by different degrees (in software).

System set-up and organization. We have closely followed the swarm robotic principles for controlling the group. Although we do not impose any necessity for local communication and interaction. The details of our control architecture has been described in Sec. 3.3. Since our robots can not localize themselves by their own hardware we provided them their instant position and orientation (*pose*) data information from a multi-robot tracking system. Sec. **??** describes about our tracking system. This system also helps us in recording and logging individual robot's task performance and communication patterns. We use Bluetooth communication link, built-in with our E-puck robots, for host PC to robot communication. Robot can also communicate with each-other physically over Bluetooth. However, we do not use that mode of communication. Instead, inter-robot communication is done virtually in host PC's D-Bus inter-process communication channel. This has been illustrated in Sec. 3.2.2. sectionHardware

3.1.4 E-puck robots

Figure 3.2: (a) The E-puck robot with SwisTrack marker on top, (b) A binary coded marker that can be tracked by an overhead camera using SwisTrack software.

We use E-puck [2] robots developed by Swiss Federal Institute of Technology at Lausanne (EPFL) and now produced by Cyberbotics [3] and some other companies. The upside of using E-puck is: it is equipped with most common sensing hardware, relatively simple in design, low cost, desktop-sized and offered under open hardware/software licensing terms. So any further modification in hardware/-software is not limited to any proprietary restriction. However, the downside of using E-puck robot is: it's processor is based on dsPIC micro-controller (lack of standard programming tool-chains), limited amount of memory (lack of on-board camera image processing option) and default communication module is based-on Bluetooth (limited bandwidth and manual link configuration).
E-puck can be programmed through C language and this program can be uploaded from PC to robot through wire: I^2C and RS232 channel or, through wireless: Bluetooth communication channel. This can be tedious and time-consuming if one needs to change the robot controller frequently. However, we intend to keep the robot's functionalities very simple and limited to two main tasks: avoiding obstacles and navigating from one place to another. Thus the default hardware of E-puck seems enough for our experiments.
Table 3.1 lists the interesting hardware information about an E-puck robot. The 7

[2]www.e-puck.org
[3]http://www.cyberbotics.com

Table 3.1: E-puck robot hardware

Feature	**Description**
Diameter	About 7 cm
Motion	Max. 15 cm/s speed (with 2 stepper motors)
Battery power	about 3 hours (5Wh LiION rechargeable battery)
Processor	16 bits micro-controller with DSP core, Microchip dsPIC 30F6014A at 60MHz (about 15 MIPS)
Memory	RAM: 8 KB; FLASH: 144 KB
IR sensors	8 IR sensors measuring ambient light and proximity of obstacles in a range of 4 cm
LEDs	8 red LEDs on a ring and 1 green LED in the body
Camera	Colour camera (max. resolution of 640x480)
Sound	3 omni-directional microphones and on-board speaker capable of playing WAV or tone sounds
Communication	Bluetooth wireless (for robot-PC & robot-robot link)

cm diameter desktop-sized robot is easy to handle. It's speed and power autonomy is also reasonable compared with similar miniature robots such as Khepera and its peers (**?**). The IR sensors provide an excellent capabilities for obstacle avoidance task. We do not make use of the tiny camera of E-puck. The combination of sound and LEDs can be very effective to detect low-battery power or any other interesting event. By default, E-puck is shipped with a basic firmware that is capable of demonstrating a set of it's basic functionalities. Using the supplied Bluetooth serial communication protocol, *BTCom* protocol, it is possible to establish serial communication link between host PC and robot firmware at a maximum possible speed of 115 kbps. From any text-based modem control and terminal emulation program, e.g. Minicom [4], one can remotely send BTCom commands to E-puck robot, e.g., set the speed of the motors, turn on/off LEDs and read the sensor values, e.g., read the IR values or capture image of the camera etc.

3.1.5 Overhead GigE camera

In order to set-up a multi-robot tracking system, we have selected a state-of-the-art GE4900C colour camera (Fig. 3.3) from Prosilica [5]. The Prosilica GE-Series

[4]http://alioth.debian.org/projects/minicom/
[5]http://www.prosilica.com

Table 3.2: Features of Prosilica GigE Camera GE4900C

Feature	**Description**
Type	Charge-coupled device (CCD) Progressive
CCD Sensor	35mm Kodak KAI-16000
Size (L x W x H)	66x66x110 (in mm)
Resolution	16 Megapixels (4872x3248)
Frame rate	Max. 3 frames per second at full resolution
Interface	Gigabit Ethernet (cable length up to 100 meters)
Image output	Bayer 8 and 16 bit

camera, are very compact, high-performance machine vision cameras with Gigabit Ethernet interface. Table 3.2 lists its main features. This GigE camera is built with CCD technology that converts light into electric charge and process it into electronic signals. Unlike in a complementary metal oxide semiconductor (CMOS) sensor, CCD provides a very sophisticated image capturing mechanism that gives high uniformity in image pixels. The 4872x3247 resolution enables us to track a relatively large area e.g., 4m x 3m. In this case, 1 pixel dot in image roughly can represent approximately 1mm x 1mm area. Although the frame rate may seem low initially, but this small frame-rate gives optimum image processing performance with large image sizes, e.g. 16 MB/frame. Prosilica offers both Windows and Linux SDK for image capture and other necessary operations. Using this SDK and OpenCV computer vision library [6], we have converted default Bayer8 format image into RGB format image and used that with our tracking software.

3.1.6 Server PC configuration

We use Dell Precision T5400 server-grade PC with the following main technical specifications: This high performance PC has supported us implementing our algorithms without having any fear of running out of resources e.g. CPU or memory. The maximum supported RAM of a 32 bit PC architecture is limited to 2 GB. But since we have used 32GB RAM in our Server PC, we have selected a 64-bit OS, Ubuntu Linux 9.10 (amd64) [7]. As an open-source Linux OS, Ubuntu offers excel-

[6]http://opencv.willowgarage.com/

[7]http://www.ubuntu.com/

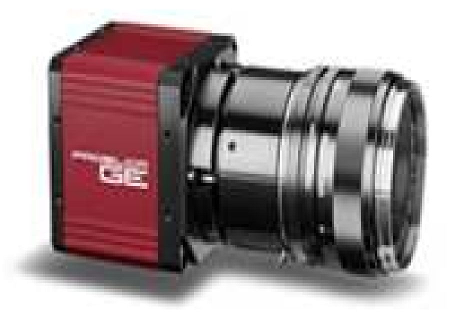


Figure 3.3: A GigE4900C camera.

Figure 3.4: A bluetooth hub that connects multiple Bluetooth adapters with Server PC.

Table 3.3: Server PC Configuration

Processor	Quad-Core Intel Xeon Processor up to 3.33GHz (1333MHz FSB, 64-bit, 2X 6MB L2 cache)
RAM	32GB (4GB ECC DIMMS x 8 slots)
Graphics Card	NVIDIA Quadro FX 570 (Memory: 256MB)
Hard-disk	SATA 3.0Gb/s 7200RPM 2 x 250 GB
OS	Ubuntu Linux 9.10 64bit

lent reliability, performance and community support. In order to enable Bluetooth communication in our host PC, we have added 8 USB-Bluetooth adapters (Belkin F8T017) through a suitable USB-Bluetooth hub (Fig. 3.4).

3.2 Enabling software tools and frameworks

3.2.1 SwisTrack: a multi-robot tracking system

In almost all types of robotic experiments, vision-based tracking becomes the standard feasible solution for tracking robot positions, orientations and trajectories. This is due to the low cost of camera hardware and availability of plenty of standard image-processing algorithms from computer vision and robotic research community. However, setting-up a real-time multi-agent tracking platform using existing software solutions are not a trivial job. Commercial systems tend to provide sub-millimetre level high precision 3D tracking solutions with a very high price ranging from 40-50 thousands of pounds which is typically greater than the annual budget of a research project! Besides robotics researchers prefer open-source solution to closed-source proprietary one due to the need for improving certain algorithms and applications continuously. Another line of solution can be borrowing certain open-source tracking code from XYZ lab. But it typically ends up with a lot of frustration while tuning parameters manually, fixing the lab lighting conditions, seeing bad performance of programs that frequently leak memory or show segmentation fault and so forth. GUI or camera calibration can hardly be found in those so-called open-source applications. The third option for solving this tracking issue becomes "re-inventing the wheel" or hiring some research students to build a system from the scratch. Certainly this is also not feasible due to the limitations in time, resource and skill needed to produce such a solution. Another big issue is the expiration of research fellowship before doing any practical research!
From the beginning of our research, we met the above types of scenario. We got very high price quotes from several commercial motion capture solution providers including, Vicon [8] and some others. We tested some well-known and some not-so-well-known open-source object tracking systems including ARTag [9], ARToolKit-

[8]http://www.vicon.com
[9]http://www.artag.net

Figure 3.5: Our experiment arena captured by Prosilica GigE4900C camera mounted on XX m high ceiling.

Figure 3.6: SwisTrack tracking a team of 16 robots under Ubuntu Linux 9.10 OS.

Plus [10]. We also developed our own versions of Open-CV algorithms for tracking colour blobs based on a GNOME application Cynbe's vison-app [11]. However, our algorithms failed to scale well due to the fluctuations in lighting conditions, lack of proper integration of all related components and some other issues (**?**). Finally, we settled with SwisTrack (**?**) , a state-of-the-art open-source multi-agent tracking platform developed at EPFL, Switzerland . Thanks to the hard working developers and generous sponsors of EPFL who have offered this excellent tool to the scientific research community and empowered many researchers to track multi-agents or multi-robots out-of-the-box.

With the improved version 4 released in February 2008, SwisTrack is now becoming one of the *de-facto* standard tool for multi-robot tracking. Being open-source, flexible, modular and customizable, Swistrack provides a clean development and deployment path for tracking marked or marker-less objects in real-time. SwisTrack is written in C++ using common C/C++ programming libraries and frameworks. The component-based modular development style is very powerful for developing a custom algorithm and wrap it in a custom component. The GUI provides a rich user interface with a clean separation between algorithmic code and parameters used in those algorithms Fig. 3.6).

In Sec. 3.2.2 we show that how we append our custom communication components in the image processing pipeline (Fig. 3.7). This pipeline can be compared with Unix command processing where output of one command becomes the input of another subsequent command and many commands form a chain or pipeline. Here in SwisTrack, at first an image capturing component grab camera image using

[10]http://studierstube.icg.tu-graz.ac.at/handheld_ar/artoolkitplus.php
[11]http://muq.org/ cynbe/vision-apps

Figure 3.7: SwisTrack image processing pipeline uses image capture, pre-process, blob-detection, tracking and other algorithms.

USB, IEEE1394/Firewire, GigE or other supported interfaces. Then subsequent SwisTrack components work on this image and do various processing e.g., background subtractions, colour conversions, blob-detection, tracking etc. These components follow standard computer-vision algorithms and can be used without any code modification. But if necessary they can also be modified or optimized though changing source code and/or tuning parameters on-the-fly in SwisTrack GUI. As shown in Fig. 3.6, this GUI can take parameters in real-time and update images. Final output from images, e.g., object position, orientation, trajectory etc. can be sent over standard communication interfaces, e.g. TCP/IP, NMEA etc. SwisTrack has a lot of other features, such as multi-camera tracking, remote-control of SwisTrack over TCP/IP etc. which are documented in SwisTrack Wiki-book documentation [12].

We have set-up SwisTrack with our Prosilica GigE camera GE4900C and configured it for tracking our E-puck robots with their on-top markers (Fig. 3.2). These markers are binary-coded numbers (aka *circular bar-codes*) that have certain binary bits or chip-lengths. We have used 20 bits bits chip-lengths. In order to uniquely identify the position and orientation of these markers, these binary numbers are encoded with a fixed hamming distance, i.e. differences in bits of any two binary numbers. We have used a fixed hamming distance of 6 bits. As shown in Fig. 3.2, these markers are 8cm diameter and clearly identified and tracked by SwisTarck from a camera image resolution of 4872x3248. SwisTrack has no component for grabbing our Prosilica GigE camera and we have developed a Prosilica GigE input component using Prosilica SDK and OpenCV library. The version of SwisTrack that we have used (May 2008 version, SVN no. XX) has worked pretty well except a few minor things, such as real-time configuration changing was very unstable due to our large camera image size (16MB/frame). In order to avoid that we put necessary configurations in SwisTrack project files that is loaded by SwisTrack in the beginning of our experiment runs.

Fig. 3.7 shows the components that we have used throughout our experiments. Along with the standard blob detection and circular bar-code reading components, we use our custom D-Bus server communication component that send pose information to D-Bus IPC channels (Sec. 3.2.2). These components require a little

[12]http://en.wikibooks.org/wiki/Swistrack

tuning of few parameters, e.g. blob size, blob counts etc. They can be done once and saved in component configuration files for loading them in next runs. Although SwisTrack provides a wide range of components for object trajectory tracking we have not used them yet. we have not saved grabbed camera images as video from within SwisTrack due to heavy CPU load and memory usage. Besides, the video output component of our version of SwisTrack can not produce smooth video files in our set-up. So, we occasionally save image frames in files and most of our experiments, we use Ubuntu Linux's standard desktop tool, *recordmydesktop* [13] for capturing screen as video. The standard fluorescent lightings set-up of our lab has seemed sufficient for our overhead GigE camera and we have prevented interferences of outside sun-lights by putting black blinds in our lab windows. Our camera has been configured automatically through standard configuration files while program start-up.

3.2.2 D-Bus: an inter-process communication protocol

Inter-process communications (IPC) among various desktop software components enable them to talk to each other and exchange data, messages or request of services. Technological advancements in computer and communication systems now allow robotic researchers to set-up and conduct experiments on multi-robot systems (MRS) from desktop PCs. Many compelling reasons, including open licensing model, availability of open-source tools for almost free of cost, community support etc., make Linux as an ideal operating system for MRS research. However the integration of heterogeneous software components in Linux desktop becomes a challenging issue, particularly when each robot-control software needs sensory and other data input from various other software components (e.g. pose data from a pose-tracker, task information from a task-server etc).

Traditional IPC solutions in a standard Linux desktop, e.g. pipes, sockets, X atoms, shared memory, temporary files etc. (hereafter called *traditional IPCs*), are too static and rigid to meet the demand of a dynamic software system (**?**). On the other hand, complex and heavy IPC like CORBA fails to integrate into development tool-chains efficiently. They also require a steep learning curve due to their complex implementations. Besides, the failure of Desktop Communication Proto-

[13]http://recordmydesktop.sourceforge.net/

col (DCOP) in system-wide integration and interoperability issues encouraged the development of the D-Bus message bus system, D-Bus for short (**?**). This message bus system provides simple mechanisms for applications to talk to one another. In this paper we describe how we exploit the simplicity and power of D-Bus for running a large MRS.

Traditional IPCs lack the important requirements of IPC among several heterogeneous software components of a large MRS. Firstly, real-time support in IPC is critical for connecting time-critical control applications. For example, a multi-robot tracking system (MRTS) can share robot pose information with a robot-controller client (RCC) though shared memory (SHM). This pose information can be used to help navigating a robot in real-time. However if MRTS crashes and stops writing new pose information into the SHM, RCC has no default mechanism to know that SHM data is outdated. Some form of reference counting mechanism can be used to overcome this issue, but that makes the implementation of RCC complicated and error-prone.

Secondly, IPC must be scalable so that adding more software components (thus more robots, sensors, etc.) in the information sharing game do not affect the overall system performance. But clearly this can not be achieved through traditional IPCs, e.g. SHM or temporary files, as the access to computer memory and disk space is costly and time consuming. Thirdly, IPC should be flexible and compatible enough to allow existing software components to join with newly developed components in the information process sharing without much difficulties. Again existing IPCs are too static and rigid to be integrated with multiple software components. Besides, incompatibility often arises among different applications written in different programming languages with different IPC semantics. Fourthly, IPC should be robust, fault-tolerant and loosely coupled so that if one ceases to work others can still continue to work without strange runtime exceptions. Finally, IPC should be implemented simply and efficiently in any modern high level programming languages, e.g., C/C++, Java, Python. Practically, this is very important since IPC will be required in many places of code and application programmers have little time to look inside the detail implementation of any IPC.

In this dissertation, we present a scalable and distributed multi-robot control architecture built upon D-Bus IPC that works asynchronously in real-time. It has virtually no limit on the number of software components who share information. By using only the signalling interfaces, SwisTrack (**?**), an open-source multi-robot tracking tool can be integrated with our multi-robot control framework. All software components are loosely coupled and unlike traditional IPCs, one does not depend on another for setting up and shutting down IPC infrastructure. For example, in case of SHM one software component explicitly needs to set-up and clean-up SHM spaces. In case of D-Bus any software component can join and leave in the information sharing process at any time. Each component implements its own fall-back strategy if desired information from another component is unavailable at any time. Based on a thin C API, D-Bus also provides many binding in common programming languages. In this work, we use *dbus-python*, a Python binding for D-Bus, that provide us a very clean and efficient IPC mechanism.

D-Bus Overview

D-BUS was designed from scratch to replace CORBA and DCOP to fulfil the needs of a modern Linux system. D-BUS can perform basic application IPC as well as it can facilitate sending events, or signals, through the system, allowing different components in the system to communicate. D-BUS is unique from other IPCs in several ways: e.g. 1) the basic unit of IPC in D-BUS is a message, not a byte stream, 2) D-BUS is bus-based and 3) It has separate system-wide and user/session-wide bus (**?**). The simplest form of D-Bus communication is process to process. However, it provides a daemon, known as the *message bus daemon*, that routes messages between processes on a specific bus. In this fashion, a bus topology is formed (Fig. 3.8). Applications can send to or listen for various events on the bus. D-Bus specification (**?**) provides full details of D-Bus message protocols, message and data types, implementation guidelines etc. Here we discuss some relevant part of this specification. Fig. 3.8 an example of DBus system structure. Here a few basic D-Bus terminologies have been introduced from D-Bus literature.

D-Bus Connection: *DBusConnection* is the structure that a program first uses to initiate talking to the D-Bus daemon, Programs can either use DBUS_BUS_SYSTEM or DBUS_BUS_SESSION to talk to the respective daemons.

DBus Message: It is simply a message between two process. All the DBus intercommunication are done using *DBusMessage*. These messages can have the fol-

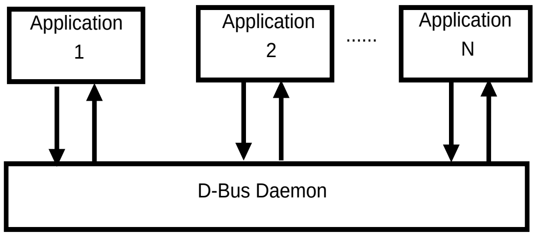


Figure 3.8: A typical view of D-Bus message bus system.

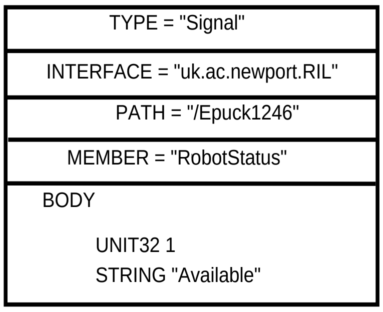


Figure 3.9: A typical structure of a D-Bus signal message.

lowing four types: method calls, method returns, signals, and errors. The DBusMessage structure can carry data payload, by appending boolean integers, real numbers, string etc. to the message body.

D-Bus Path: This is the path of a remote *Object* (capitalized to avoid ambiguity) of target process, e.g. /org/freedesktop/DBus.

D-Bus Interface: This is the interface on a given Object to talk with, e.g. org.freedesktop.DBus.

D-Bus Method Call: This is a type of DBus message that used to invoke a method on a remote Object.

D-Bus Signal: This is a type of DBus message to make a signal emission. Signal messages must have three header fields: PATH giving the object the signal was emitted from, plus INTERFACE and MEMBER giving the fully-qualified name of the signal. Fig. 3.9 shows the design of robot-status signal that emits over specified interfaces and paths with a data payload of an integer and a string containing robot-status message.

D-Bus Error: This is the structure that holds the error code which occurs by calling a DBus method.

Strategies for Application Integration

Under D-Bus, there are two basic mechanisms for applications to interact with each other: 1) by calling a remote Object of target application and 2) by emitting a signal for interested applications. To perform a method call on a D-BUS Object, a method call message must be sent to that Object. It will do some processing and return either a method return message or an error message. Signals are different in that they cannot return anything: there is neither a "signal return" message, nor any other type of error message [14]. Thus on D-Bus everything can be done asynchronously without the need of polling.
D-Bus provides several language bindings for integrating D-Bus to any native application. The core D-BUS API, written in C, is rather low-level and large. Bindings integrate with programming languages and environments, e.g. Glib, Python, Qt and Mono. The bindings provide environment-specific features. For example, the Glib bindings treat D-BUS connections as *GObjects* and allow messaging to integrate into the *Glib mainloop*. The preferred use of D-BUS is definitely using language and environment-specific bindings, both for ease of use and improved functionality (**?**).

3.2.3 BTCom/Myro: E-puck robot control programs

E-puck robot comes with a set of software tools and libraries to program and to monitor low-level sensor and actuator values. The low-level C library with driver code of E-puck robot can be downloaded from E-puck website [15]. In order to modify and recompile this library E-puck developers recommend both Windows and Linux cross-compling tool-chains. Under Windows, the MPLAB environment from Microchip [16] can be used with their C30 compiler. We have used this commercial tool for programming E-puck robot (dsPIC micro-controller), since the Linux counterpart, piklab [17] has been found unstable and still under-development. A wide variety of boot-loaders, both under Windows and Linux, can be used to upload the *.hex* firmware files to E-puck robot over Bluetooth. We have found a most

[14]http://www.ibm.com/developerworks/linux/library/l-dbus.html
[15]www.e-puck.org
[16]http://www.microchip.com
[17]http://piklab.sourceforge.net/

reliable E-puck boot-loader built-in with the trial version of Webots [18] simulator. E-puck website also provides various other tools, e.g. Player robot control framework driver [19] Matlab interfacing program, E-puck-monitor (under Windows) etc. for monitoring (or setting) sensors (or actuator) values. The default firmware running in E-puck robot is *BTCom*. It initializes Epuck robot hardware and waits for user command over Bluetooth serial port. A set of well-defined BTCom commands can be found in Epuck-library documentation.

For high-level control of E-puck robot, we have used Myro robot-control framework [20] that is developed by Institute for Personal Robots in Education. While BTCom provides a set of user commands for controlling the robot, it does not take care the setting up the Bluetooth serial connection. Moreover the supplied user commands are very primitive in nature. For example, from a high-level perspectives it is more desirable to command a robot for moving at a specific speed for a given time. Under BTCom, a user can not achieve this without interactively giving low-level motor commands, e.g. set left/right motor speed. On the other hand, by setting up Myro framework, the Bluetooth communication with host PC and E-puck robot can easily be set-up under Python's pySerial [21] module. This provides an elegant solution for controlling E-puck from software code. Besides, Myro provides a thin wrapper code for E-puck's BTCom for defining high-level user commands. For example, instead of setting left/right motor speed, a user can send a forward command with speed and time-out as its parameters. With the simplicity and interactivity of Python programming, this wrapper makes E-puck programming and debugging very simple and easy.

The default BTCom has another limitation that it's detection of low battery voltage is almost unnoticeable by naked eye. For running a long time experiment this is critical since we would like to continue our experiments even if a few robots' batteries run out. By the default code of BTCom, a tiny red LED, located near the poer LED in E-puck body, turns on when battery voltage becomes low. This LED light is not visible from a crowd of robots. In order to overcome this issue we modified BTCom so that it can turn of all LEDs when battery voltage becomes

[18]http://www.cyberbotics.com/products/webots/
[19]http://code.google.com/p/epuck-player-driver/
[20]http://wiki.roboteducation.org/
[21]http://pyserial.sourceforge.net/

critical. This exploited the hardware interrupt signal from Low-Voltage-Detection (LVD) module of E-puck hardware.
Finally we developed our custom navigation and obstacle avoidance algorithms in Python. The navigation function is based on our camera pose information. In each time-step, robot gets it current pose information from our multi-robot tracking system. It then determines its current coordinate (location) relative to the target object and calculates the differences in pose and orientation. To advance forward, it at first corrects its heading based on the difference and then moves forward for a small fixed distance towards the target. Of course, in every time-step, it also checks that if it is located within the target object's boundary and if this is the case, it ceases its motion. Obstacle avoidance algorithm works under the navigation code. While a robot tries to move forward if an obstacle is sensed by its IR sensor it makes a random turn and tries to avoid it. Due to the noisy sensor values, it takes two or a few time-steps to completely get rid of that obstacle.

3.2.4 BlueZ: Linux's Bluetooth communication stack

The physical communication between the host PC and E-puck robot occurs over Bluetooth wireless radio communication channel. As defined by the Bluetooth Special Interest Group's official technology info site [22]

> "*Bluetooth* technology is a wireless communications technology intended to replace the cables connecting portable and/or fixed devices while maintaining high levels of security. The key features of Bluetooth technology are robustness, low power, and low cost".

The obvious reason for selecting Bluetooth as the communication technology of E-puck is perhaps due to it's low cost, low battery usage and universality of hardware and software. Each E-puck robot has a Bluetooth radio link to connect to a host PC or nearby other E-puck robots. Under the hood, this Bluetooth chip, LMX9820A [23], is interfaced with a Universal Asynchronous Receiver/Transmitter (UART) microchip of E-puck robot. This Bluetooth chip can be used to access to the UART "transparently" using a bluetooth rfcomm channel. Using this mode, one can access the e-puck as if it is connected to a serial port. According to the

[22]www.bluetooth.com
[23]http://www.national.com/opf/LM/LMX9820A.html

specification of LMX9820A, it supports Bluetooth version 1.1 qualification, This means that the maximum supported data transfer speed is 1Mbit/s. But typically it is configured to use a serial port's 115200 bits/s speed.

Before starting to use an E-puck robot one needs to set-up the Bluetooth connection with the robot. Typical Bluetooth connection set-up from a Bluetooth-enabled host PC includes a few manual steps: detecting the remote Bluetooth device, securely bonding the device (e.g. exchanging secret keys) and setting-up the target rfcomm or serial connection (over radio) channel. Various Bluetooth software stacks are available under different OSes. Under Linux, BlueZ [24] becomes the *de-facto* standard software platform. The BlueZ stack was initially developed by Max Krasnyansky at Qualcomm [25] and in 2001 they decided to release it under the GPL. The BlueZ kernel modules, libraries and utilities are known to be working prefect on many architectures supported by Linux. It offers full support for Bluetooth device scanning, securely pairing with devices, rfcomm or serial link configuration, monitoring and so forth. Initial scanning and secure bonding of E-puck devices can be done by a set of BlueZ tools, namely, *hcitool, l2ping, hciconfig, rfcomm* etc. While initializing, BlueZ's core daemon, *bluetoothd,* reads the necessary configuration files (e.g. rfcomm.conf) and dynamically sets up or binds all Bluetooth devices' links and thereafter, routes all low-level communications to them. BlueZ's supplementary package, Hcidump, offers logging raw data of all Bluetooth communications over a host PC's Bluetooth adapter. In our host PC, we have used USB dongle type Bluetooth adapter. We have also used various Linux serial connectivity tools, e.g *minicom, picocom etc.* to test link configurations and to send BTCom commands to E-puck robots.

From the above points, we can see that setting up and maintaining connectivity to E-puck robots through Bluetooth links is not a trivial task. Thus, one needs to consider automating the process of Bluetooth link set-up and verification in order to save time in initializing real experiments. Moreover, comparing with other common wireless technologies, e.g. Wifi, Bluetooth is a relatively low-bandwidth technology. In case of almost all wireless technologies, presence of lots of wireless devices causes significant noises and interferences. Thus, one also needs to con-

[24]www.bluez.org

[25]http://www.qualcomm.com/

sider the channel capacity or total available bandwidth for communications. Within the context of our experiments, we have got an interesting open question. *What is the maximum number of E-puck robots that can talk to host PC simultaneously ?*. However, finding the answer of this question is beyond the scope of this thesis and here we would only like to stick with the feasible configuration of Bluetooth links without modifying any low-level protocols or technical implementation.

3.2.5 Python's Multiprocessing: process-based multi-threading

The real-time interactions among multiple software applications often require concurrency and synchronization, to some degrees, in their functions. Although a common inter-process communication (IPC) protocol, e.g. D-Bus, solves the problem of data-sharing among different application processes, synchronization of data in various processes remains a challenging issue. The idea of simultaneous and parallel execution of different part of application codes without any IPC, typically on multiple CPU cores, introduces the notion of *multi-threading* programming. Both process-based and thread-based approach of program execution has pros and cons. Threads are light weight and they can share memory and state with the parent process without dealing with the complexity of IPC. Threads can be useful to the algorithms which rely on shared data/state. They can increase throughput by processing more information faster. They can also reduce latency and improve the responsiveness of an application, such as GUI actions. However, since threads implicitly "share everything" programmers have to protect (lock) anything which will be shared between threads. Thus thread-based programs are subject to face race conditions or deadlocks among multiple threads.
On the other hand, processes are independent process-of-control and they are isolated from each other by the OS. In order to do any data/state sharing they must use some form of IPC to communicate and coordinate. Comparing with threads, processes are big and heavy since process creation takes time and these processes also tends to be large in program size and memory footprint. Since processes "share nothing" – programmers must explicitly share any data/state with suitable mechanism. From a high-level robotic programmer's point of view, both thread-based and process-based application design approaches are disadvantageous. Since thread-based approach requires careful attention in data-sharing it becomes very difficult

to design bug-free program in short time-scale. On the other hand process-based approach requires to set-up IPC mechanisms and manage them. However, the latter approach is less likely to produce bugs as data sharing is explicit.

Almost all modern computer OSes and *high-level* programming languages, e.g. C/C++, Java etc. offer multi-threading support. However implementation of multi-threading programming involves lots of low-level thread management activities. In this respect *very high-level* programming languages e.g. Python, Ruby etc. offer more efficient and elegant solutions for dealing with multi-theaded programs. Along with this multi-threading issue, various other factors influence us to use Python for coding our high level robotic programs. For example, Python programs are usually optimized through *byte-codes* (like Java programs), and then they are *interpreted* by various Python interpreters. Unlike dealing with compiling issues in most of the high-level programming languages, Python allows programmers to focus on their algorithms more quickly and integrate their systems more effectively. We have found Python's interactive program development process more productive and flexible than non-interactive programming approach found in some other languages.

Starting from from version 2.6, Python offers an integration of thread-based programming with process-based programming through it Multiprocessing module [26]. Traditionally, Python offers threads that are real, OS/Kernel level POSIX *pthreads*. But, older Python programs can have only a single thread to be executing within the interpreter at once. This restriction is enforced by the so-called "Global Interpreter Lock" (GIL). This is a lock which must be acquired for a thread to enter the interpreters space. This limits only one thread to be executing within the Python interpreter at once. This is enforced in order to keep interpreter maintenance easier. But this can also be sidestepped if the application is I/O (e.g. file, socket) bound. A threaded application which makes heavy use of sockets, will not see a huge GIL penalty. After doing a lot of research on various alternatives of this approach, Python community has offered Multiprocessing as a feasible solution to side-step GIL by the CPU-bound applications that require seamless data/state sharing. It follows the threading API closely but uses processes and IPC under the hood. It also offers distributed-computing facilities as well, e.g. remote data-

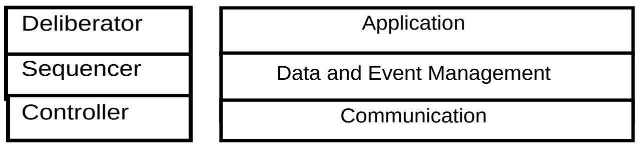
[26]http://docs.python.org/library/multiprocessing.html

sharing and synchronization. Thus, we have exploited the power and efficiency of Multiprocessing that enables us to make a modular and flexible implementation of multi-robotic software system. Sec 3.3 explains some of our implementations of Python Multiprocessing module.

Python Multiprocessing offers various mechanisms for sharing data among processes or, more precisely speaking, among sub-processes. We have used a separate *Manager* process that handles all the data storage tasks and event-based process synchronizations. Managers are responsible for network and process-based sharing of data between processes (and machines). The primary manager type is the *BaseManager* that is the basic Manager object, and can easily be subclassed to share data remotely. We use this Multiprocessing Manager object that runs a server process in one machine and offers data objects through proxies in parallel to many client processes over network interfaces.

3.3 Multi-robot control architecture

As defined by (**?**), a robot control architecture is "a set of guiding principles and constraints for organizing a robot's control system". The overall aim of a multi-robot control architecture is to tie various necessary software components that enable a group of robots to work together and to achieve a common goal, such as self-regulated MRTA, by following a set of guiding principles and constraints. Since our self-regulated MRTA solution closely follows the bio-inspired swarm robotic system's paradigm, we have selected simple E-puck robots. Under this paradigm, we have chosen distributed self-organized task-allocation approach (Chapter 4) that requires each robot to run its own task-allocation algorithm independently for selecting and switching among tasks (recall from Sec. 2.4 and Fig. 2.41). Robots need to gather task related informations to run these task-allocation algorithm. Moreover, BTCom commands, that autonomously drive the robots, are also sent from a host PC's robot controller client program. Robots are also need real-time pose data from multi-robot tracking system. All these requirements indicate us that we need a suitable multi-robot control architecture that can effectively tie all these heterogeneous software components together. But how can we do that ? In this Section, we have answered this question by presenting a multi-robot control archi-



(a) Three layers (b) Abstract model of HEAD

Figure 3.10: (a) Classical three-layer hybrid robot control architecture after **?** (b) Our abstract multi-robot control architecture adopted from hybrid architecture.

tecture, Hybrid event-driven architecture on D-Bus (*HEAD*). As discussed in Sec. **??**, this architecture uses D-Bus IPC mechanism for providing real-time, scalable, fault-tolerant and efficient interactions among various software components.

3.3.1 Hybrid event-driven architecture on D-Bus (*HEAD*)

As we have discussed in Sec. 2.3 robotic researchers have spent a lot of efforts for finding suitable robot control architecture. Since last few decades, robot control architectures has been evolving from deliberative to reactive and hybrid (combination of deliberative and reactive), behaviour-based and to some other forms. It has been well established that hybrid control can bring together the best aspects of both reactive and deliberative control by combining the real-time low-level device control and high-level deliberative action control. Only reactive (or deliberative) control approach is not enough for enabling robots to do complex tasks in a dynamic environments (**?**).

As shown in Fig. 3.10(a), hybrid control is usually achieved by a three-layer architecture composed of deliberator, sequencer and controller . *Controller* usually works under real-time reactive feedback control loops to do simple tasks by producing primitive robot behaviours, e.g. obstacle avoidance, wall following etc. *Deliberator* performs time-consuming computations, e.g. running exponential search or computer vision algorithm processing. In order to achieve specific task goals, the middle component, *sequencer*, typically integrates both deliberator and controller maintaining consistent, robust and timely robot behaviours.

Three layers of HEAD

We have organized our multi-robot control architecture, HEAD into three layers as shown in Fig. 3.10(b). Although HEAD has been designed by adopting the principles of hybrid architecture it has many distinct features that are absent or overlooked in a classical hybrid architecture. Firstly, with respect to controller layer, HEAD broadly views sensing and control as communication with external entities. Communication as sensing is not new, e.g. it has been reported in multi-agent learning (**?**). When robots' on-board computing resources are limited communication can effectively make up their required sensing capabilities. On the other hand, low-level device control is also a series of communication act where actuator commands are typically transmitted over a radio or physical link. Thus, all external communication takes place at the *communication layer*. Components sitting in this layer either act as sensors that can receive environmental state, task information, self pose data etc. via suitable communication link or do the real-time control of devices by sending actuator commands over a target communication channel. For example, in this study robot controller clients receive pose data from multi-robot tracking system through this communication layer. Similarly, from a host-PC, robot controller clients send E-puck robots' actuator commands over Bluetooth wireless radio to physical robots.

Secondly, the apparent tight coupling with sensors to actuators has been reduced by introducing a *data and event management (DEM) layer*. DEM layer acts as a short-term storage of sensor data and various events posted by both controller and deliberator components. Task sequencing has been simplified by automated event triggering mechanism. DEM simply creates new event channels and components subscribe to their interested event channels for reading or writing. If one components updates an event, DEM notifies subscribed components about this event. Controller and deliberator components synchronize their tasks based on this event signals. DEM efficiently serves newly arrived data to the controller and deliberator components by this event sharing mechanism. For example, upon receiving robot pose data, our robot controller clients save this data and notifies to other components about the availability of pose data so that they can do their work using this updated pose data. Thus unlike traditional hybrid architecture, neither specialized languages are needed to program a sequencer nor cumbersome if/else checks are

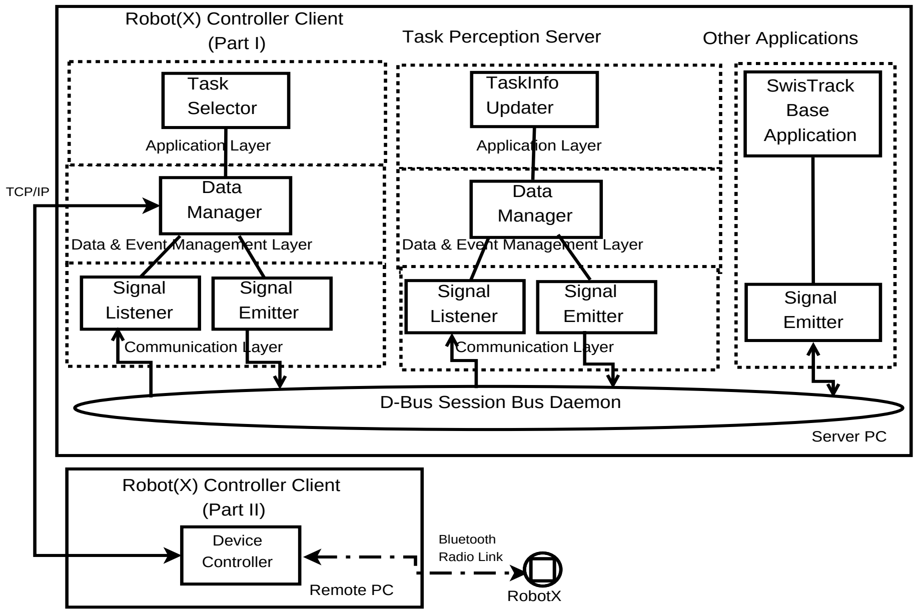


Figure 3.11: General outline of *HEAD*. A RCC application has been split into two parts: one runs locally in server PC and another runs remotely, e.g., in an embedded PC.

present in this layer.

Finally, deliberator layer of HEAD has been described as an *application layer* that runs real-application code based on high-level user algorithms as well as low-level sensor data and device states. For example, our self-regulated MRTA algorithms have been fitted in this layer. In classic hybrid architecture the role of this layer has been described mainly in two folds: 1) producing task plan and sending it to sequencer and 2) answering queries made by sequencer. Application layer of HEAD follows the former one by generating plan and queuing it to DEM layer, but it does not support the latter one. DEM layer never makes a query to an application since it acts only as a passive information gateway. Thus this reduced coupling between DEM and application layer has enhanced HEAD with additional robustness and scalability. Additional applications can be added with DEM layer's existing or new event interfaces. Any malfunction or failure in application layer or even in communication layer can be isolated without affecting others.

3.3.2 Software component integrations

Fig. 3.11 outlines the placements of various software components based on their functional characteristics and processing requirements. Here we have discussed how these software components can be integrated in the communication layer of HEAD using D-Bus IPC. The exact implementation of each of these components are left to be discussed in the following chapters within our specific context of MRTA applications. Note that, here we use the term *software component* or *application* to denote the logical groupings of several sub-processes or threads that works under a mother process or main thread (here we use the term thread and process interchangeably). Software components that follows our three-layer architecture for grouping its processes are called *native component* whereas existing software applications are called *external component*. As shown in Fig. 3.11, robot-controller client (RCC) and task-information provider, task perception server (TPS) are native software components of HEAD, whereas SwisTrack (**?**), an external tool used with HEAD, is called an external component.

In order to integrate both native and external components with HEAD. We have designed two separate communication process: a D-Bus signal reception process, *SignalListener*, and a D-Bus signal emission process, *SignalEmitter*. Inside a native component both of this process can communicate with data and event management process, DataManager, by using any suitable mechanisms, such as, multi-threading, multi-processing (offered in Python multiprocessing as discussed in Sec. 3.2.5), TCP or any other networking protocol.

Any external component that intend to act as a sensing (actuating) element of HEAD need to implement a SignalEmitter (SignalListener). For example, we extend SwisTrack with D-Bus signal emitting code (aka SignalEmitter) so that it can emit robot pose messages to individual robots D-Bus path under a common interface. This emitted signal is then caught by SignalListener of individual robot's RCC. Thus the tight-coupling between SwisTrack and RCC has been removed. During run-time SwisTrack can flexibly track variable number of robots and broadcast their corresponding pose messages without any re-compilation of code. Moreover, in worse cases, if SwisTrack or RCC crashes it does not affect any other component at run-time.

Expanding SignalEmitter and SignalListener for more D-Bus signals does not require to make any change the IPC implementation code. Steps for setting up a SignalEmitter is listed below.

Step 1: Connect to a D-Bus daemon.

Sample C code:

```
DBusError error;
DBusConnection *conn;
dbus_error_init (&error);
/* Get a connection to the session bus */
conn = dbus_bus_get (DBUS_BUS_SESSION, &error);
```

Step 2: Optionally, reserve a D-Bus path or service name (this is not required if the same B-Bus path is not used by any other process).

Step 3: Send a signal to a specified path.

Sample C code:

```
DBusMessage *message;
message = dbus_message_new_signal("/target/dbus/path",
"target.dbus.interface","SignalData");
/* Send the signal */
dbus_connection_send (connection, message, NULL);
dbus_message_unref (message);
```

Steps for setting up signal reception:

A signal can be received by setting up a suitable event loop under any supported language bindings. This event loop include a special callback function that become activated when this signal received properly. For example, the callback function of a robot-pose signal can save the pose data into memory or files upon receiving it. This event loop often includes another error handling function that become activated when an error is occurred while receiving the signal. Using a Glib *mainloop* interface [27], the simplified steps of a typical SignalListener is listed below.

Step 1: Set-up a Glib event-loop.

Sample C Code:

```
/* glib main loop */
GMainLoop *loop;
loop = g_main_loop_new(NULL,FALSE);
```

Step 2: Connect to a D-Bus daemon (same as above).

Step 3: Add a match for the target D-Bus signal

Sample C Code:

[27]http://library.gnome.org/devel/glib/

```
/* D-bus signal match */
dbus_bus_add_match (connection,
"type='signal',interface='target.dbus.interface'",NULL);
dbus_connection_add_filter (connection,
 dbus_signal_callback, loop, NULL);
```

Step 4: Set-up DBus-Glib call.

Sample C Code:

```
/* dbus-glib call */
dbus_connection_setup_with_g_main(connection,NUL);
```

Step 5: Run Glib event-loop.

Sample C Code:

```
/* run glib main loop */
g_main_loop_run(loop);
```

In the above listing `dbus_signal_callback` function contains the specific application code that determines what to do with the received signal (which is not shown here). `dbus_connection_add_filter` function can take error handing function as its last argument (or can be NULL as shown here). D-Bus signal match rules are specified in D-Bus specification (**?**). In both of the above cases, i.e. signal listening and receiving, in order to add more signals we just need to repeat step 3 as many times as we need. A basic implementation of both of signal emission and listening processes in Python language can be found in this here [28]. In Chapter 4 Chapter 5, we have discussed these Pythonic way of D-Bus signal handing within the context of our MRTA applications.

[28]http://dbus.freedesktop.org/doc/dbus-python/doc/tutorial.html

CHAPTER 4

Validation of Attractive Field Model for Self-regulated MRTA

4.1 Motivations

In this chapter we have discussed about the attractive field model (AFM) (**?**) an interdisciplinary model of self-regulated task-allocation or division of labour (DOL) in social systems. The model has been developed under the EPSRC collaborative project “Defying the rules how self-organizing systems work”. Here, we have studied three different social systems: ants, humans and robots in order to identify generic mechanisms that lead sustainability of social systems through self-regulation [1].

A trans-disciplinary study

The idea of finding a generic model of DOL, by collaboratively studying these social systems, has many fascinating advantages.
Firstly, this interdisciplinary study makes it possible to develop a generic model of self-regulatory DOL by combining the strengths of different disciplines overcoming their individual shortcomings. For example, ant colonies are the ideal example for studying the self-regulatory social systems, however it is very difficult to pin-

[1]The partners of this project were from University of West of England, University of Hull, University of Wales, Newport and Imperial College London and they researched on ants, humans, robots and mathematical models respectively

point the exact mechanism leading to a specific behaviour (**?**). Artificial systems e.g. MRS can be used to explore and verify biological hypotheses using totally controlled experiments. Similarly observational data from human social systems can be combined with the data from the biological experiments to enhance our understanding of social self-regulatory mechanisms.

Secondly, synergy of different methods, experimental and observational data from disparate disciplines gives us the ability to construct a more abstract, yet powerful, model which may not be available through independent studies. The higher-level abstraction can be very helpful to increase the usefulness of this model since we can easily separate generic and the domain-specific components of the model. Generic part guides us to design a core-framework that can be used to create basic characteristics of a system, whereas domain-specific part can be implemented independent of other disciplines. For example,different social systems use different communication mechanisms, yet almost all of them share many common aspects in their self-regulatory behaviours (see Sec. 2.2 for an example).

Thirdly, the scope of application of our generic models has been extended in many folds by tackling the common challenges of different disciplines. For example, both human social organizations and multi-robot systems suffer from the scalability issue for large organization. The properties of local sensing and local communication with relatively incapable sensory organs/hardware are present in both ants and swarm robotic systems. Thus, the integration of solutions from three major disciplines gives us a highly flexible and extensible model of DOL.

The construction of AFM has been achieved through a series of collaborative interactions among all project partners. From the biological experiments of ants colonies *Temnothorax albipennis* we inferred the bottom-level rules and roles of feedback in collective performance of ants brood-sorting and nest construction after emigration to a new site. We also analysed the observational data from the self-organized infrastructural development of an *eco-village* by an open community of volunteers resided in Ireland. These helped us to identify the generic rules of DOL in social systems. Later on, we formalized these generic rules into AFM (**?**) and validated them by putting AFM into our robot controllers within the context of a virtual manufacturing shop-floor scenario. In this chapter, we have described mainly the later part of our study, i.e. robotic validation of AFM, with a brief presentation on interpretations of AFM from different social perspectives.

4.2 The Attractive Field Model (AFM)

4.2.1 Generic framework

Inspired from the DOL in ants, humans and robots, we have proposed the following four rules that are the necessary ingredients to obtain self-regulation in any social system. In this dissertation, these rules are mentioned as *generic rules of self-regulation*.

Rule 1: Continuous flow of information. Self-regulatory systems need to establish the continuous minimum flow of information over the period of time, where self-regulation can be defined. This should maintain at least two states of an agent: 1) receiving information about task(s) and 2) ignoring information or doing no task. The updated information should reflect the changes of the system i.e. it will encode the necessary feedback for the agents. Thus, this property will act as the basis of the smooth switching of states, between these two minimum states or, among multiple states (e.g. in case of multiple tasks or many sub-states of a single task) .

Rule 2: Sensitization. Self-regulatory systems allow the differentiation in the use of (or access to) information, e.g. through sensitization or learning some tasks. This differentiation is regulated by the characteristics of the system, e.g. the ability of the agents to learn tasks that are repeatedly performed.

Rule 3: Concurrence. Self-regulatory systems include concurrent access to information from different spatial positions with certain preferences. This preference is not fixed and can change with the dynamics of the system.

Rule 4: Forgetting. Self-regulatory systems include forgetting, e.g. the ability of the agents to diminish information over time, if not used. The system determines the amount of information being released, and this changes over time. For example, specialists might have to attend an emergency situation and switch tasks that contributes to the forgetting of old task experiences. This is considered as crucial to allow flexibility in the system.

Having this general framework of self-regulation, we can now formalize AFM that will describe the properties of individual agents and the system as a whole. In terms of networks, the model is a bipartite network, i.e. there are two different types of nodes. One set of nodes describes the sources of the attractive fields and the other set describes the agents. Links only exist between different types of nodes and they encode the flow of information so that, even if there is no direct link between two agents, their interaction is taken into account in the information flow. This is an instance of *weak* interaction. The strength of the field depends on the distance between the task and the agent. This relationship is represented using weighted links. In addition, there is a permanent field that represents the *no-task* or option for ignoring information. The model can be mapped to a network as shown in Fig. 4.1. The correspondence is given below:

1. Source nodes (o) are tasks that can be divided between a number of agents.

2. Agent nodes (x) an be ants, human, robots etc.

3. The attractive fields correspond to stimuli to perform a task, and these are given by the black solid lines.

4. When an agent performs a task, the link is of a different sort, and this is denoted in the figure by a dashed line. Agents linked to a source by a red line are the agents currently doing that task.

5. The field of ignoring the information (w) corresponds to the stimulus to random walk, i.e. the no-task option, and this is denoted by the dotted lines in the graph.

6. Each of the links is weighted. The value of this weight describes the strength of the stimulus that the agent experiences. In a spatial representation of the model, it is easy to see that the strength of the field depends on the physical distance of the agent to the source. In addition, the strength can be increased through sensitisation of the agent via experience (learning). This distance is not depicted in the network, it is represented through the weights of the links . In the figure of the network, the nodes have an arbitrary place. Note that even though the distance is physical in this case, the distance in the model applied to other systems, needs not to be physical, it can represent the accessibility to the information, the time the information takes to reach the receiver, etc.

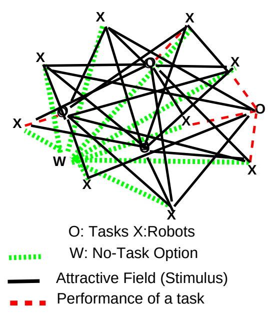


Figure 4.1: Attractive Filed Model (AFM)

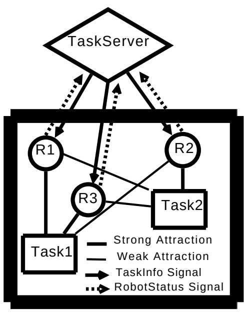


Figure 4.2: A centralized communication scheme

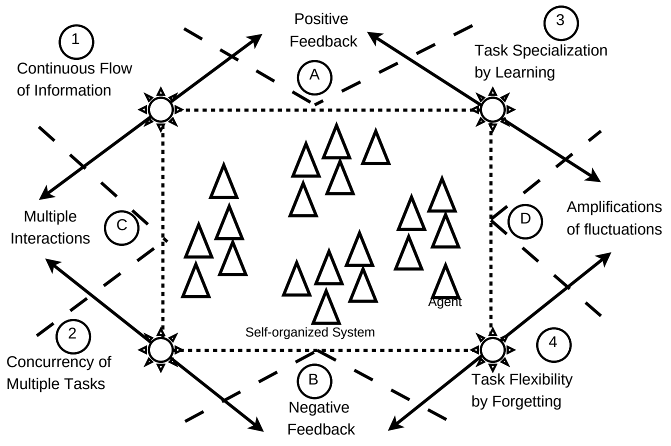

1
Continuous Flow of Information
Positive Feedback
A
3
Task Specialization by Learning
Multiple Interactions
C
D
Amplifications of fluctuations
Agent
Self-organized System
2
Concurrency of Multiple Tasks
B
Negative Feedback
4
Task Flexibility by Forgetting

Figure 4.3: Four generic rules establish the self-regulated DOL in social systems

In summary, looking at the network, we can see that each of the agents is connected to each of the fields. This means that even if an agent is currently involved in a task, the probability that it stops doing it in order to pursue a different task, or to random walk, is always non-zero. The weighted links express the probability of an agent to be attracted to each of the fields.

4.2.2 Relationship of AFM with self-organization

It is interesting to note that our proposed four generic rules has become the four major corners of the foundation of a self-organized system (Fig. **??**). As discussed in Sec. **??**, self-organized systems exhibit four distinct perspectives known as so-called ingredients or properties of self-organization. However, it is not clear how those properties can come into existence. Here, we describe the four underlying mechanisms that explains how self-organization can be realized in different social systems. From our understanding, we can explain it in the following ways.

Firstly, multiple interaction becomes meaningful when *continuous flow of information* occurs by exchanging signals or cues among agents or their environment that regulates their behaviours. This, in turn, contribute to the task-allocation and task-switching in the social level. In the biological and swarm-intelligence (SI) literature, multiple interactions are often described as an essential ingredient of self-organization. However, interactions without definite purposes may not contribute to the self-organization.

Secondly, in SI, positive feedback has been attributed as another mechanism of self-organization. But it is not easy to understand what creates positive feedback in a social system. Possible answers might be the characteristic of the environment (e.g., ants select shorter path since density of pheromones becomes higher and thus more ants becomes attracted in that path), the decrease of response-threshold of individuals (thus increase of probability of selecting a task) etc. To make the answer more concrete, we have explicitly attributed *sensitisation* or learning as a mechanism of positive feedback. There might exist other mechanisms too. But clearly sensitisation will be one of the reliable mechanisms for achieving positive feedback.

Thirdly, similar to positive feedback, we have proposed *forgetting* that contributes to provide negative feedback about a task or decreasing the probability to select it. Other negative feedback mechanisms can be implemented by assigning a saturation level to each task which is also present in our model, for details see **?**.

Finally, creating artificial amplification of fluctuations or stochastic events is not a straight-forward issue. It throws many open questions. Does a system designer intentionally impose irregularity in task-performance of agents ? Is random movement enough for simulating randomness in a system ? Since emergencies do not always pop-up on request, we provide the rule of *concurrency* that enables agents to maintain even a small amount of probability of selecting a low-priority or less sensitized or distant task. This concurrency mechanism provides a high-degree of robustness in the system such that all tasks can be attended even if specialization of agents delays them in switching to some of the tasks.

4.2.3 Interpretation of AFM in an ant colony

The interpretation of AFM in an ant colony almost exactly follows the above generic interpretation. Moreover it also reveals a few additional characteristics of AFM. For example, at the individual level, information is processed differently by each individual, and is certainly not constant nor continuous. In addition, there can be lower and upper thresholds on the amount of info necessary for DOL to take place. The time scale at the individual level is very small compared to the system level's time scale.We can approximate the propagation of information at this macro time scale as the continuous flow of information. In the model, emphasize is given to whether the information is used e.g. stimulation to perform a task, or unused e.g. random walk (RW).

For ants we have assumed that division of labour is not genetically driven. Initially they are all equal. It is not fully understood how ants "know" or get information about tasks. But we know that they all interact directly and indirectly and perform tasks. The flow of information can take place in many different ways. As we have discussed in Sec. 2.2, ants can get information through direct P2P, local or global broadcast or indirect pheromone communication. In any case, if an ant i_1 is closer to the source of information, i.e. task, than any other ant i_2, there will be larger probability that i_1 will get that information than i_2. If all ants are assumed to be initially equal it will be more likely that i_1 will attend to that task. Thus each task can be treated as an attractive source, stimulating ants to go to it. Here the stimulus primarily depends on the distance.

In case of sensitization or learning, an ant that has performed a task, it is assumed that it will be more likely that it will be attracted to do it again. Here there is no encoding of individual performance, the specialists are more attracted towards the task but do not perform the task quicker than other ants. Thus the performance of colony increases as a result of sensitisation. Simultaneity (concurrence) of tasks over period of time of self-regulation can also be achieved through spatial dependence of strength of stimulus. Some ants will be favoured to get information from multiple sources than others. When an ant does not do a task for relatively long time it is less probable that it will do that task again. Thus, Flexibility in task switching is achieved through forgetting. A concrete interpretation of AFM in an ant colony, along with simulation results can be found in (**?**).

4.2.4 Interpretation of AFM in a human society

The interpretation of AFM in a human society can be made using many different approaches. For example the following can be mapped to different nodes and characteristics of AFM in a human society.

- Source nodes (o) can be resources.
- Agents (x) can be people.
- The links correspond to the flow of information, resources, etc.
- The distance dependence can be introduced here in the same way as with the ants. People working in a certain area are more likely to receive info/resources with respect to that area, than another person doing something else. How the person uses that info/resources, corresponds to the people's ability to contribute towards a given goal.The weight of the link is the amount of info a person gets. The person can use it or dismiss it. Random walking would correspond to dismiss it. AFM does not differentiate between good and bad info, it is always considered as good info.

Similar to the above biological interpretation, we can see that in human society the division of labour can be also interpreted in terms of AFM. People can get information about certain tasks or resources from various sources. Those who are located near the source of information are more likely to attend it, e.g. through the access to Internet, some people can get information quicker than others. This can be treated as the distance to tasks or resources. The motivation of a person to use information can also be treated as the distance in the model. The background training, education or skill profile can be used to estimate the sensitization to do a task or use a resource. The urgency of a task can be inferred through some other estimates, e.g. the frequency of mentioning about a task or resource, by team members or peers, can indicate its relative urgency.

4.2.5 Interpretation of AFM in a MRS

The interpretation of AFM in a MRS also follows almost exactly as in generic interpretation. However, in order to make the interpretation more concrete, let us consider a manufacturing shop floor scenario, where N number of autonomous mobile robots are required to attend J number of shop tasks spread over a fixed area A. Let these tasks be represented by a set of small rectangular boxes resembling to manufacturing machines.

Let R be the set of robots $r_1, r_2, ..., r_n$. Let a task j has an associated task-urgency ϕ_j indicating its relative importance over time. If a robot attends a task j in the x^{th} time-step, the value of ϕ_j will decrease by an amount $\delta_{\phi_{INC}}$ in the $(x+1)^{th}$ time-step. On the other hand, if a task has not been served by any robot in the x^{th} time-step, ϕ_j will increase by another amount $\delta_{\phi_{DEC}}$ in $(x+1)^{th}$ time-step. Thus urgency of a task is updated by the following rules.

$$If\ the\ task\ is\ not\ being\ done:\ \phi_j \rightarrow \phi_j\ + \delta_{\phi_{INC}} \tag{4.1}$$

$$If\ the\ task\ is\ being\ done:\ \phi_j \rightarrow \phi_j\ - n\ \delta_{\phi_{DEC}} \tag{4.2}$$

Eq. 4.1 refers to a case where no robot attend to task j and Eq. 4.2 refers to another case where n robots are concurrently performing the task j.

In order to complete a task j, a robot r_i needs to be within a fixed boundary D_j. If a robot completes a task j it learns about it and this will influence r_i's likelihood of selecting that task in future, say through increasing its sensitization to j by a small amount, k_{INC}. Here, the variable affinity of a robot r_i to task j is called as its *sensitization* k_j^i. If a robot i does not do a task j for some time, it forgets about j and k_j^i is decreased, by another small amount, say k_{DEC} . Thus a robot's task-sensitization update follows these rules.

$$If\ task\ is\ done:\ k_j^i \rightarrow k_j^i\ +\ k_{INC} \tag{4.3}$$

$$If\ task\ is\ not\ done:\ k_j^i \rightarrow k_j^i\ -\ k_{DEC} \tag{4.4}$$

According to AFM, all robots will establish attractive fields to all tasks due to the presence of a system-wide continuous flow of information. The strength of these attractive fields will vary according to the dynamic distances between robots and tasks, task-urgencies and corresponding sensitizations of robots. Simplifying the generic implementation of AFM from **?**, we can formally encode this stimuli of attractive field as follows.

$$S_j^i = tanh\{\frac{k_j^i}{d_{ij}+\delta}\phi_j\} \tag{4.5}$$

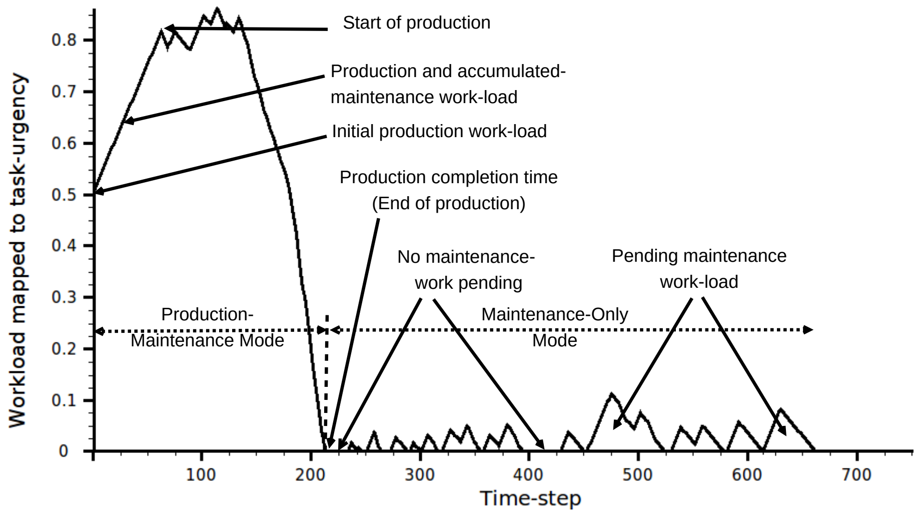


Figure 4.4: Virtual Shop-floor production and maintenance cycle

$$S_{RW}^{i} = tanh\left\{1 - \frac{\sum_{j=1}^{J} S_j^i}{J+1}\right\} \tag{4.6}$$

$$P_j^i = \frac{S_j^i}{\sum_{j=0}^{J} S_j^i} \quad where, \quad S_0^i = S_{RW}^i \tag{4.7}$$

Eq. 4.5 states that the stimuli of a robot r_i to a particular task j, S_j^i depends on r_i's spatial distance to j (d_{ij}), level of sensitization to j (k_j^i), and perceived urgency of that task (ϕ_j). In Eq. 4.5, we have used a very small constant value δ to avoid division by zero, in the case when a robot has reached to a task. Since S_j^i is a probability function, it is chosen as a $tanh$ in order to keep the values between 0 and 1. Eq. 4.6 suggests us how we can estimate the stimuli of random walk or no-task option. This stimuli of random walk depends on the sum of stimulus of J real tasks. Here, random-walk is also considered as a task. Thus the total number of tasks become $J + 1$. The probability of selecting each task has been determined by a probabilistic method outlined in Eq. 4.7 which states that the probability of choosing a task j by robot r_i is directly proportional to its calculated stimuli S_j^i. Finally, let T_a be the allocated time to accomplish a task. If a robot can enter inside the task boundary within T_a time it waits there until T_a elapsed. Otherwise it will select a different task.

4.3 A mock manufacturing shop-floor (MMS)

By extending our interpretation of AFM for MRS from Sec. 4.2, we can set-up a mock manufacturing shop-floor (MMS) scenario. Here, each task represents a manufacturing machine that is capable of producing goods from raw materials, but they also require constant maintenance works for stable operations. Let W_j be a finite number of material parts that can be loaded into a machine j in the beginning of its production process and in each time-step, ω_j units of material parts can be processed ($\omega_j \ll W_j$). So let Ω_j^p be the initial production workload of j which is simply: W_j/ω_j unit. We assume that all machines are identical. In each time step, each machine always requires a minimum threshold number of robots, called hereafter as *minimum robots per machine (μ)*, to meet its constant maintenance work-load, Ω_j^m unit. However, if μ or more robots are present in a machine for production purpose, we assume that, no extra robot is required to do its maintenance work separately. These robots, along with their production jobs, can do necessary maintenance works concurrently. For the sake of simplicity, here we consider μ = 1.

Now let us fit the above production and maintenance work-loads and task performance of robots into a unit task-urgency scale. Let us divide our manufacturing operation into two subsequent stages: 1) *production and maintenance mode (PMM)*, and 2) *maintenance only mode (MOM)*. Initially a machine starts working in PMM and does production and maintenance works concurrently. When there is no production work left, it then enters into MOM. Fig. 4.4 illustrates this scenario for a single machine. Under both modes, let α_j be the amount of workload occurs in a unit time-step if no robot serves a task and it corresponds to a fixed task-urgency $\Delta\phi_{INC}$. On the other hand, let us assume that in each time-step, a robot, i, can decrease a constant workload β_i by doing some maintenance work along with doing any available production work. This corresponds to a negative task urgency: $-\Delta\phi_{DEC}$. So, at the beginning of production process, task-urgency, occurred in a machine due to its production work-loads, can be encoded by Eq. 4.8.

$$\Phi_{j,INIT}^{PMM} = \Omega_j^p \times \Delta\phi_{INC} + \phi_j^{m0} \tag{4.8}$$

where ϕ_j^{m0} represents the task-urgency due to any initial maintenance work-load of j. Now if no robot attends to serve a machine, each time-step a constant maintenance workload of α_j^m will be added to j and that will increase its task-urgency by $\Delta\phi_{INC}$. So, if k time steps passes without any production work being done, task urgency at k^{th} time-step will follow Eq. 4.9.

$$\Phi_{j,k}^{PMM} = \Phi_{j,INIT}^{PMM} + k \times \Delta\phi_{INC} \tag{4.9}$$

However, if a robot attends to a machine and does some production works from it, there would be no extra maintenance work as we have assumed that μ = 1. Rather, the task-urgency on this machine will decrease by $\Delta\phi_{DEC}$ amount. If ν_k robots work on a machine simultaneously at time-step k, this decrease will be: $\nu_k \times \Delta\phi_{DEC}$. So in such cases, task-urgency in $(k+1)^{th}$ time-step can be represented by:

$$\Phi_{j,k+1}^{PMM} = \Phi_{j,k}^{PMM} - \nu_k \times \Delta\phi_{DEC} \tag{4.10}$$

At a particular machine j, once $\Phi_{j,k}^{PMM}$ reaches to zero, we can say that there is no more production work left and this time-step k can give us the *production completion time* of j, T_j^{PMM}. Average production time-steps of a shop-floor with M machines can be calculated by the following simple equation.

$$T_{avg}^{PMM} = \frac{1}{M}\sum_{j=1}^{M} T_j^{PMM} \tag{4.11}$$

T_{avg}^{PMM} can be compared with the minimum number of time-steps necessary to finish production works, T_{min}^{PMM}. This can only happen in an ideal case where all robots work for production without any random walking or failure. We can get T_{min}^{PMM} from the total amount of work load and maximum possible inputs from all robots. If there are M machines and N robots, each machine has Φ_{INIT}^{PMM} task-urgency, and each time-step robots can decrease N $\times$ $\Delta\phi_{DEC}$ task-urgencies, then the theoretical T_{min}^{PMM} can be found from the following Eq. 4.12.

$$T_{min}^{PMM} = \frac{M \times \Phi_{INIT}^{PMM}}{N \times \Delta\phi_{DEC}} \tag{4.12}$$

$$\zeta_{avg}^{PMM} = \frac{T_{avg}^{PMM} - T_{min}^{PMM}}{T_{min}^{PMM}} \tag{4.13}$$

Thus we can define ζ_{avg}^{PMM}, *average production completion delay* (APCD) by following Eq. 4.13: When a machine enters into MOM, only μ robots are required to do its maintenance works in each time step. So, in such cases, if no robot serves a machine, the growth of task-urgency will follow Eq. 4.9. However, if ν_k robots are serving this machine at a particular time-step k^{th} , task-urgency at $(k+1)^{th}$ time-step can be represented by:

$$\Phi_{j,k+1}^{MOM} = \Phi_{j,k}^{MOM} - (\nu_k - \mu) \times \Delta\phi_{DEC} \tag{4.14}$$

By considering $\mu = 1$, Eq. 4.14 will reduces to Eq. 4.10. Here, $\Phi_{j,k+1}^{MOM}$ will correspond to the *pending maintenance work-load (PMW)* of a particular machine at a given time. This happens due to the random task switching of robots with a no-task option (random-walking). Interestingly PMW will indicate the robustness of this system since higher PMW value will indicate the delay in attending maintenance works by robots. We can find the average PMW (APMW) per time-step per machine, χ_j^{MOM} (Eq. 4.15) and average PMW per machine per time-step, χ_{avg}^{MOM} (Eq. 4.16).

$$\chi_j^{MOM} = \frac{1}{K}\sum_{k=1}^{K} \Phi_{j,k}^{MOM} \tag{4.15}$$

$$\chi_{avg}^{MOM} = \frac{1}{M}\sum_{j=1}^{M} \chi_j^{MOM} \tag{4.16}$$

4.4 Experiment design

We have designed a set of virtual manufacturing shop-floor (MMS) experiments for validating the effectiveness of our attractive field model (AFM) in producing self-regulated multi-robot task allocation (MRTA). The overall aim of this design is to analyse the various properties of task-allocation and related other issues. In this section, we have described the design of the observables and parameters of our experiments within the context of our MMS scenario.

4.4.1 Observables

Plasticity: As we have discussed in Sec. 2.1.3, self-regulated division of labour or task-allocation can be characterised by plasticity and task-specialization, in both macroscopic and microscopic levels. Within MMS context, plasticity refers to the collective ability of the robots to switch from doing no-task option (random-walking) to doing a task (or vice-versa) depending on the work-load present in the system. Here we expect to see that most of the robots would be able to engage in tasks when there would be high workloads (or task-urgencies) during production and maintenance mode (PMM). Similarity, when there would be low workload in case of maintenance-only mode (MOM) only a few robots would do the task, rest of them would either be idle (not doing any task) or perform a random-walk. The changes of task-urgencies and the ratio of robots engaged in tasks can be good metrics to observe plasticity in MRTA.

Task-specialization: Under heavy work-load most of the robots should attend to tasks. But self-regulated division of labour is always accompanied with task-specializations of agents. That means, few robots will be more active than others. From AFM, we can see that after doing a task a few times, a robot will soon be sensitized to it. Therefore, from the raw log of task-sensitization of robots, we can find the pattern of task-sensitization of robots per task basis. If a few robots specializes on a particular task that will help to reduce traffic near the task and improves overall efficiency of the system. Thus, at the end of our production cycle in MMS scenario, we can count the percentage of robots specializes on each task in our experiments.

Quality of task-performance: As discussed in Sec. 4.3 we can measure the quality of MRTA from the average production completion delay (APCD). APCD first calculate the ideal minimum production time and then find the delay in production process from the actual production completion data. Thus this will indicate how much more time is spent in the production process due to the self-regulation of robots in this distributed task-allocation scheme. In order to calculate APCD, we can find the production completion time for each task from the raw log of task-urgency and make an average from them.

Robustness: In order to see if our system can respond to the gradually increasing workloads, we have measured average pending maintenance work (APMW) within the context of our MMS scenario. This can prove the robustness of our system where a task can be unattended for long time. When a task is not being served by any robot for some time we can see that its urgencies will rise and robots will respond to this dynamic demand. For measuring APMW we need only the task-urgency data.

Flexibility: From the design of AFM, we know that robots that are not doing a task will be de-sensitized to it or forget that task. So at an overall low work-load (or task urgency), less robots will do the tasks and hence less robots will have the opportunity to learn tasks. From the robot sensitization data, we can confirm the presence of flexibility in MRTA.

Energy-efficiency: In order to characterize the energy-efficiency in MRTA we also log the pose data of each robot that can give us the total translations occurred by all robots in our experiments. This can give us a rough indication of energy-usage by our robots.

Information flow: Since AFM requires a system-wide continuous flow of information, we can measure the communication load to bench-mark our implementation of communication system. This bench-mark data can be used to compare among various communication strategies (Chapter 5). Here we measure the how much task-related information, i.e. task-urgency, location etc. are sent to the robots at each time step. This amount of information or communication load can be constant or variable depending on the design of the communication system. **Scalability:** In order to see the effects of scaling on MRTA, we have designed two series of experiments. *Series A* corresponds to a small group where we have used 8 robots, 2 tasks under an arena of 2 m^2. We have doubled these numbers in Series B, i.e. 16 robots, 4 tasks under an arena of 4 m^2. This proportional design can give us a valuable insight about the effects of scaling of our system on MRTA under the AFM. All of the above metrics of Set A and Set B can be compared to find those scalability effects.

Thus, in order to observe the above properties of self-regulated MRTA , we have designed our experiments to record the following observables in each time-step.

1. Task-urgency of each task (ϕ).
2. Number of robots engaged in each task.
3. Task-sensitizations (k) of robots.
4. Pose data of robots.
5. Communication of task-information message with robots.

Table 4.1: Experimental parameters

Parameter	Series A $\mid$ Series B
Total number of robots (N)	8 $\mid$ 16
Total number of tasks (M)	2 $\mid$ 4
Experiment area (A)	2 m^2 $\mid$ 4 m^2
Initial production work-load/machine (Ω_j^p)	100 unit
Task urgency increase rate ($\Delta\phi_{INC}$)	0.005
Task urgency decrease rate ($\Delta\phi_{DEC}$)	0.0025
Initial sensitization (K_{INIT})	0.1
Sensitization increase rate (Δk_{INC})	0.03
Sensitization decrease rate (Δk_{DEC})	0.01

4.4.2 Parameters

Table 4.1 lists a set of essential parameters of our experiments. We intend to have a setup that is relatively complex, i.e., with a high number of robots and tasks in a large area. The diameter of the marker of our e-puck robot is 0.08m. So, if we put 4 robots in an area of one square meter, this will give us a robot-occupied-space to free-space ratio of about 1:49 per square meter. This ratio reasonable in order to allow the robots to move at a speed of 5 cm/sec without much interference to each other.

The initial values of task urgencies correspond to 100 units of production work-load without any maintenance work-load as outlined in Eq. 4.8. We choose a limit of 0 and 1, where 0 means no urgency and 1 means maximum urgency. Same applies to sensitisation, where 0 means no sensitisation and 1 means maximum sensitisation. This also implies that if sensitization is 0, task has been forgotten completely. On the other hand, if sensitization is 1, the task has been learnt completely. We choose a default sensitization value of 0.1 for all tasks. The following relationships are maintained for selecting task-urgency and sensitization parameters.

$$\Delta\phi_{INC} = \frac{\Delta\phi_{DEC} \times N}{2 \times M} \tag{4.17}$$

$$\Delta k_{DEC} = \frac{\Delta k_{INC}}{M - 1} \tag{4.18}$$

Eq. 4.17 establishes the fact that task urgency will increase at a higher rate than that of its decrease. As we do not like to keep a task left unattended for a long

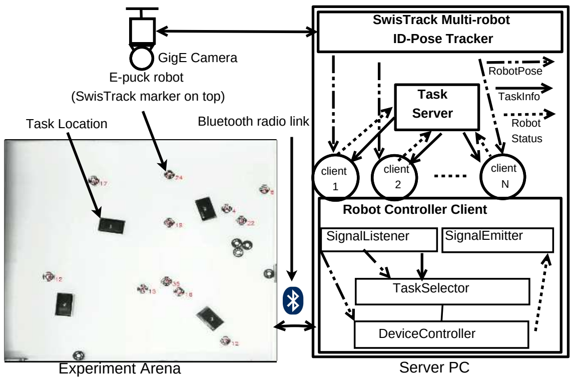


Figure 4.5: Hardware and software setup

time we choose a higher rate of increase of task urgency. This difference is set on the basis of our assumption that at least half of the expected number of robots (ratio of number of robots to tasks) would be available to work on a task. So they would produce similar types of increase and decrease behaviours in task urgencies. Eq. 4.18 suggests that the learning will happen much faster than the forgetting. The difference in these two rates is based on the fact that faster leaning gives a robot more chances to select a task in next time-step and thus it becomes more specialized on it.

4.5 Implementation

Ideally, AFM can be implementation as a complete distributed task-allocation system where each agent selects its own task based on its own external perception about task-urgencies (i.e. attractive fields), distances from tasks and internal task-sensitisation records. Such an implementation requires powerful robots with sophisticated sensors (camera, laser etc.) and sufficient computation and communication capabilities. In such cases, robots can keep task-urgency information up-to-date through suitable local communication schemes with their peers who can monitor the tasks. By using suitable navigation and mapping modules, they can also accurately calculate the distances from tasks and navigate to tasks autonomously. Moreover, they also require necessary hardware to do the actual task, e.g. gripper for pick-up tasks.

However, in this study, we are particularly interested to find the suitable communication schemes that can effectively spread the attractive fields (task-urgencies) among robots. So we have simplified the complexities of a full-fledged implementation by using a centralized communication system (CCM) that effectively makes up the limitations of our robots. For example, our robots are not capable of sensing a task to estimate its urgencies, instead our centralized *task-perception server (TPS)* broadcast task information, e.g. task-urgencies, locations etc. to robots in certain time intervals. Within this interval, if some robots work on a task, they independently send their status, i.e. which task they are currently doing. From this status message, TPS can re-calculate the task-urgencies and send them in next broadcast. Fig. 4.2 illustrates this by showing three robots attracted to two different tasks and their communications with TPS. Here although the robots are selecting task independently based-on the strength of their attractive fields to different tasks, they are depended on the TPS for task-information.

Certainly, this centralized communication system can be converted into a decentralized one where robots can use local observation and communication with peers about tasks to estimate task-urgencies. In Chapter 5, we present an emulation of this scenario where robots do not depend entirely on TPS for estimating task-urgencies, instead they get task information from TPS when they are very close to a task (inside a predefined task-boundary) or from local peers who know about a task via TPS.

In our current implementation, instead of doing any real work with powerful robots, we emulate a mock manufacturing shop-floor scenario that requires the robot only to travel among tasks. As discussed in Sec. 3.2.3, our robots do not have on-board CPU, they need a host PC to control them using BTCom communication protocol. Thus our host PC runs one robot-controller client (RCC) per robot. These RCCs also rely upon SwisTrack multi-robot tracking system for updating their real-time pose. So although our MRTA solution is distributed by design, we primarily used a centralized approach to implement it due to the limitations of our robots and the convenience of our implementation. Below we describe the actual implementations of these components, i.e. D-Bus communication interfaces and detail implementations of TPS and RCC.

4.5.1 D-Bus communication interfaces

Our centralized communication system interfaces among SwisTrack, TPS and RCCs as shown in Fig. **??**. The communication protocol is based on D-Bus IPC. As we have discussed in Sec. 3.2.2, each message is a D-Bus signal (similar to Fig. 3.9). The characteristics of these messages are discussed below.

RobotPose: SwisTrack emits this D-Bus signal and its payload contains individual robot's pose data: robot-pose x-coordinate, y-coordinate and robot orientation (theta). The type of each data is `DBUS_TYPE_DOUBLE`(IEEE 754 double). We have extended SwisTrack by adding a D-Bus signal emitter module which emits this signal in every image-processing cycle. The typical duration of each cycle varies from 1 to 2 seconds. Swistrack extracts robot-pose from the image frame considering top-left corner of image as origin and it derives theta from the relative position of the monochrome robot-marker from the image. SwisTrack's id-pose detection algorithms handles the complexities of finding the accurate robot-pose. RobotPose signal is sent to RCC using robot-id based separate D-Bus paths. e.g. /robot1, /robot2 etc. But all RobotPose signals are emitted in same D-Bus interface "uk.ac.newport.ril.SwisTrack". In order to get pose data, each RCC filters this D-Bus signal using this interface and self-id based path.

TaskInfo: TPS emits this signal after a fixed time intervals. The payload of this message contains an array of all task's information. Each element of the array contains a task's: task-id, Time-stamp, pose x, pose y, task-urgency (ϕ). Task-id data is integer type and all others are double. Time-stamp in data ensures using up-to-date task information since RCCs receive TaskInfo from TPS asynchronously. If the tasks are dynamic their pose can be obtained from SwisTrack as well. In our current implementation, we have provided static pose data for all task as an argument to TPS module. So the dynamic part, i.e. the time-stamp and up-to-date task-urgency is determined by the task-information-updater module of TPS, according to Algorithm **??**. This D-Bus signal is sent in a common D-Bus interface, "uk.ac.newport.ril.TaskServer", under path, "/taskserver". Each RCC receives this message by listening to this interface and path.

RobotStatus: Each RCC emits this D-Bus signal and the payload contains: individual robot's id and its currently executing task's id. This message is sent to a common D-Bus interface "uk.ac.newport.ril.Epuck" and path "/robot". But TPS

Table 4.2: Data structures and events used by the RCC

Data structure	**Related event(s)**
mRobotID	-
mRobotPose	mRobotPoseAvailable
mTaskInfo	mTaskInfoAvailable
mSelectedTask	mSelectedTaskAvailable mSelectedTaskStarted mTaskTimedOut

identifies the sender from the message pay-load, i.e. robot-id. A RCC emits this signal only when it starts working on a particular task.

4.5.2 Robot Controller Client (RCC)

As shown in Fig. 4.5, each e-puck robot is controlled by a corresponding RCC. A RCC sends commands to the robot's firmware using BTCom protocol (Sec. 3.2.3). A RCC consists of several Python modules. Each module represents a sub-process under a main process that ties all of them together by using Python's Multiprocessing module. Below we have described the detail design and implementation of these Python modules. All code are publicly accessible through GitHub repository[2].

DataManager

As we have discussed in Sec. 3.2.5, under a process-based design of IPC among multiple processes, a programmer needs to specify explicitly which data and states should be shared among sub-processes. According to the design of our multi-robot control architecture, HEAD (Sec. 3.3), we have employed a data and event management process called *DataManager* that acts as a data warehouse and event management centre for RCC.

Table 4.2 list DataManager's major data structures and corresponding event channels. Here mRobotID, an integer type data, represents the e-puck robot's marker ID (converted to decimal number from the binary code) which uniquely identifies a robot from others. This is given as a command-line argument during RCC start-up.

[2]git://@github.com/roboshepherd/EpuckCentralizedClient.git, *hash:*7e3af8902e64db3fa59e

We have used Python *dictionary* type data structure for all other data structures that are managed by the Python Multiprocessing's *Manager()* object. Any other process e.g. TaskSelector, can access all of the DataManger's data structures and event channels locally by taking a reference of this DataManager object from the Multiprocessing's parent process. DataManager object also runs a tiny TCP/IP server process powered by the Multiprocessing's *RemoteManager()* interface. So, if necessary, any other process e.g. DeviceController, can access all of the DataManger's data structures and event channels remotely by instantiating a proxy client that connects to this embedded TCP/IP server of DataManager and fetch data from it. In the following paragraphs, under the discussion of other RCC processes, the update process of rest of the data structures and event channels are explained.

SignalListener

As mentioned in Sec. 3.3.2, we have employed two D-Bus communication modules in RCC: SignalListener and SignalEmitter, that are responsible for listening and emitting D-Bus signals respectively. SignalListener, has subscribed to D-Bus session bus for listening to D-Bus RobotPose and TaskInfo signals that are discussed above. SignalListener uses two call-back functions that handles both of these signal reception events. For example when SwisTrack emits RobotPose signal the corresponding handler function becomes activated and receive this RobotPose siganl's payload data (i.e. x, y , theta). It then makes a copy of these data to DataManager's mRobotPose data structure and label them as dictionary key/value pairs, where x, y, theta etc. becomes the dictionary keys. In addition to copying the data, SignalListener also set the DataMager's mRobotPoseAvailable event to *True*. In consequence, those processes that are waiting for mRobotPose data, e.g. TaskSelector, finishes their waiting and try to access this newly arrived mRobotPose data. After reading to this data they marks this mRobotPoseAvailable event as *False* so that this pose data can be treated as outdated and waiting processes can wait for new pose update. This strategy ensures the consistent and real-time data read/write by all subscribing processes.

SignalEmitter

Similar to the SignalListener, SignalEmitter makes use of Python *sched* module that enables it to check periodically mSelectedTaskStarted event and emit a robot's task status containing signal, RobotStatus. This is done by subscribing to the mSelectedTaskStarted event channel. When a robot starts executing a task, this event channel becomes activated by the DeviceController (discussed below). At a very close time, SignalEmitter gets this event update and finishes waiting for this event. It then picks up the mSelectedTask data from DataManager and prepare the RobotStatus signal and emits it to the session bus. Upon a successful emission event, it clears mSelectedTaskStarted event so that only one RobotStatus signal is emitted per task-cycle.

TaskSelector

TaskSelector works in the application layer of our multi-robot control architecture, HEAD. It plugs the AFM algorithms into RCC to select task based-on DataManger's event notifications , i.e. mRobotPoseAvailable and mTaskInfoAvailable and upon running task-allocation algorithm it updates mSelectedTask (and mSelectedTaskAvailable) with selected task information. **Algorithm: Self-regulated task-allocation based on AFM**

```
1: Input: $AllTaskInfo, RobotPose, TaskRecords, DeltaDistance$
2: Output: $SelectedTaskID$
3: comment: Stage 1: Get each task's individual stimuli and sum all stimulus
4: $TotalTasks \leftarrow$ Total number of tasks $\in AllTaskInfo$
5: $TaskStimuliSum \leftarrow 0$
6: for all $Task \in AllTaskInfo$ do
7:     $TaskPose \leftarrow$ Task-position $\in Task$
8:     $TaskUrgency \leftarrow$ Task-urgency $\in Task$
9:     $TaskSensitization \leftarrow$ Task-sensitization $\in Task$
10:    $DistanceToTask \leftarrow$CalculateTaskDistance($RobotPose, TaskPose$)
11:    $TaskStimuli \leftarrow$ CalculateTaskStimuli($DistanceToTask,$
12:                        $TaskSensitization, TaskUrgency, DeltaDistance$)
13:    $TaskStimuliSum \leftarrow TaskStimuliSum + TaskStimuli$
14:    $TaskRecords \leftarrow$ UpdateTaskRecords($TaskStimuli,$
15:                        $DistanceToTask, TaskUrgency, TaskSensitization$)
16: end for
17: $RandomWalkStimuli \leftarrow$ CalculateRandomWalkStimuli($TotalTasks,$
18:                        $TaskStimuliSum$)
19: $AllStimuliSum \leftarrow TaskStimuliSum + RandomWalkStimuli$
20: comment: Stage 2: Find probability of each task based on its stimuli
21: $TaskID \leftarrow 0$
22: while $TaskID \leq TotalTasks$ do
23:     $TaskStimuli \leftarrow$ Task-stimuli $\in TaskRecords(TaskID)$
```

```
24:     $TaskProbability \leftarrow$ **GetTaskProbability(**$TaskStimuli, AllStimuliSum$**)**
25:     $TaskProbabilityRange \leftarrow$ **ConvertTaskProbabilityIntoRange(**
26:                                                 $TaskProbability$**)**
27:     $TaskRecords \leftarrow$ **UpdateTaskRecords(**$TaskProbability$**)**
28: end while
29: comment: Stage 3: Draw a random-number to match with TaskID
30: $RandomNum \leftarrow$ GetRandomNumber($0$, Max($TaskProbabilityRange$))
31: while $TaskID \leq TotalTasks$ do
32:     $RangeStart \leftarrow$ Min($TaskProbabilityRange(TaskID)$)
33:     $RangeEnd \leftarrow$ Max($TaskProbabilityRange(TaskID)$)
34:     if $RandomNum \geq RangeStart \;\&\&\; RandomNum \leq RangeEnd$ then
35:         $SelectedTaskID \leftarrow TaskID$
36:     end if
37: end while
```

TaskSelector allocates a robot with a shop-task or random-walk task from several inputs: as shown in Algorithm 0.Here AllTaskInfo and RobotPose correspond to the mTaskInfo and mRobotPose of DataManager. TaskRecords is a dictionary type data structure that keep tracks of related calculations, e.g. stimulus, of all tasks. This is internally maintained by the TaskSelector process. DeltaDistance is a very small constant number to avoid division by zero in calculating task stimuli, as mentioned in Eq. 4.5.

Task-allocation algorithm:

Stage 1. In the beginning of our task-allocation algorithm, we count the total number of tasks and initialize the sum of the stimuli all tasks to zero. Then for each task, we extract its pose, urgency, sensitisation from input data. CalculateTaskDistance() function calculates the Euclidian distance between a task and a the robot by taking the current robot-pose and static task-pose as the input. These values are enough to get a task's stimuli based on Eq. 4.5. In this loop finally we update TaskRecords for using it in next task-allocation cycle. We get the random-walk task's stimuli from Eq. 4.6. It requires total number of shop-tasks and sum of their stimulus.

Stage 2. In the second stage of our task-allocation algorithm, we find the probability of each task (including random-walk) based on Eq. 4.7. Then this probability value of each task, between 0 and 1, is rounded to the closest two-digit fractions and multiplied by 100 and put into a linear scale. For example, if two tasks probability values are: 0.15 and 0.25, the probability range of first task becomes 0 to 15 and for second task, it becomes 16 to 40.

Stage 3. After converting each task's probability values into a linear range, we use a random-number generator that draws a random-number between 0 and the highest value of task-probability range, say 40 for the previous example. Then this random number is compared against the task probability range of each task. For example, if the random-number becomes 37, our task-probabiliy range checking selects task2 as 37 falls between 16 to 40. Under this probabilistic method, the task with larger probability range has a higher chance to be selected, but low probability tasks are also got selected time-to-time.

Algorithm: Robot's learning and forgetting of tasks based on AFM

```
1: Input: $TaskRecords, LeranRate, ForgetRate, SelectedTaskID$
2: Output: Updated $TaskSensitisation \in TaskRecords$
3: for all $Task \in TaskRecords$ do
```

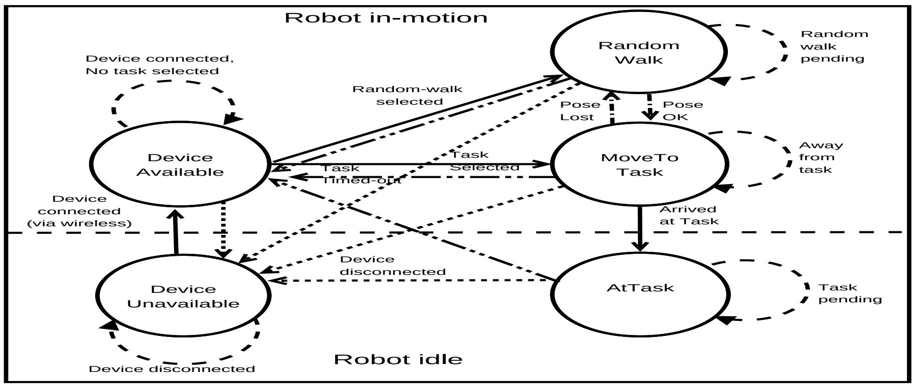


Figure 4.6: State diagram of DeviceController module

```
 4:     $TaskID \leftarrow$ ID $\in Task$
 5:     $TaskSensitization \leftarrow$ Sensitisation $\in Task$
 6:     if $TaskID \equiv SelectedTaskID$ then
 7:         $TaskSensitization \leftarrow$ Max($1, (TaskSensitization + LeanRate)$)
 8:     else
 9:         $TaskSensitization \leftarrow$ Min($0, (TaskSensitization - ForgetRate)$)
10:     end if
11: end for
```

Robot learning and forgetting algorithm:

Algorithm 0 lists the pseudo-code for updating the robot's task-sensitization values. Along with previously mentioned TaskRecords, and SelectedTaskID, it takes two other inputs: LeranRate (Δk_{INC}) and ForgetRate (Δk_{INC}) as outlined in Eq. eqn:k-inc and eqn:k-dec respectively. In addition to that it also keep the value of task-sensitization between the fixed limit of 0 and 1.

DeviceController:

The final code that moves a robot to desired task location or make random-walk resides in DeviceController module. This is also a separate sub-process of our Python Multiprocessing based mother process. It waits for the DataManager's mRobotPoseAvailable and mSelectedTaskAvailable events. When TaskSelector sets mSelectedTaskAvailable event, DeviceController sub-process checks the Bluetooth link-connectivity with physical robot. In case of successful task start-up it sets mSelectedTaskStarted event that, in turn, triggers SignalEmitter to emit a D-Bus signal publishing the robot's currently engaged task. The full state switching policies of DeviceController is illustared in Fig. 4.6. Externally we can observe two distinct physical state of a robot: it is either idle or in motion (separtated by dotten line). These two physical states are mapped into several logical steps:

DeviceUnavailable: Initially, DeviceController sets it state as DeviceUnavailable (e.g. not connected with Bluetooth link) and after a fixed short-time intervals it checks whether the device has connected to host-PC, i.e. after the wireless link becomes up. In that case, DeviceController switches its state to DeviceAvailable (*Device connected*). We can see that from every other state, device can come to this unavailable state (shown by dotted arrow) when the link is lost e.g. in case of robot's low battery or any other disconnection event (*Device disconnected*).

DeviceAvailable: DeviceController stays in this state until DataManager's mSelectedTaskAvailable event fires (*Device connected, no task selected*). Then it switches to either MoveToTask or RandomWalk state depending on the selected task. DeviceController returns to this state from RandomWalk, MoveToTask or AtTask states after a pre-set task time-out period has elapsed. In that case it triggers the DataManager's mTaskTimedOut event.

RandomWalk: Robot can random walk in two cases: 1) when TaskSelector selects this random-walk task or 2) when the robot can not get its pose information for a moment. The latter helps the pose tracker to recover the robot-pose from a crowd of robots. Robot continues to do random-walk until it's task time-out period elapses or it regains it pose remains unavailable from the pose tracker (*random-walk pending*).

MoveToTask: In this state, robot takes step-by-step navigation approach to reach a task boundary. Until the robot reaches the task boundary (*Away from task*), in every navigation-step it adjusts its heading to task based on both robot and task pose information and then makes a fixed-time translation movement. In worse cases when time-out happens early before reaching a task DeviceController switches from this state to DeviceAvailable state. However, if the same task is selected immediately, it is more likely that it will go to next AtTask step. **AtTask:** In this state DeviceController discovers itself neat the task and normally until task time-out happens (*Task pending*) it stays at this state.

4.5.3 Task perception server (TPS)

Similar to RCC, TPS has two D-Bus communication components: SignalListener and SignalEmitter, and a data and event management component DataManager

(Fig. 3.11). This instance of DataManager stores statistics about task performance of robots in a dictionary a type data structure (mTaskWorkers). This data structure is updated every time a RobotStatus signal is received from a RCC. It is a dictionary type data structure where ID of tasks are keys and ID of robots are the values of this dictionary. Based on this data structure, an application layer component, TaskInfoUpdater, updates the task-urgencies after every short time intervals. This update algorithm in presented in Algorithm 0. Along with the static task-pose information, all task-urgencies are stored in the mTaskInfo data structure of DataManager of TPS. This TPS's SignalEmitter waits for the update of mTaskInfo and after every certain intervals it emits this up-to-date TaskInfo signal.

Algorithm: Task-urgency update rules based on AFM

```
1:  Input: $AllTaskInfo, AllTaskWorkers, ThresholdWorkers,$
2:         $DeltaUrgencyINC, DeltaUrgencyDEC$
3:  Output: Updated $TaskUrgencies \in AllTaskInfo$
4:  for all $(Task, Workers) \in (AllTaskInfo, AllTaskWorkers)$ do
5:      $TaskUrgency \leftarrow$ Task-urgency $\in Task$
6:      $TaskWorkers \leftarrow$ Number of workers $\in Workers$
7:      if $TaskWorkers < ThresholdWorkers$ then
8:          $TaskUrgency \leftarrow$ Max($1, (TaskUgerncy + DeltaUrgencyINC)$)
9:      else
10:         $TaskUrgency \leftarrow$ Min($0, (TaskUgerncy -$
11:                                  $TaskWorkers \times DeltaUrgencyDEC)$)
12:     end if
13: end for
```

This algorithm acts upon the task-urgency data of each task and updates it based on the active number of robots currently working on that task (Eq. 4.1 and 4.2). Here AllTaskInfo and AllTaskWorkers correspond to mTaskInfo and mTaskWorkers of TPS DataManager respectively. ThresholdWorkers are the minimum number of robots required to work on this task (μ). DeltaTaskUrgencyINC and DeltaTaskUrgencyDEC are the task-urgency increase and decrease rate which correspond to the small increase and decrease of work-load in every-step. This algorithm also keeps the values of task-urgencies within 0 and 1 limit by using generic Min() and Max() functions. Source-code of a Python implementation of TPS is publicly accessible through GitHub repository[3].

4.6 Results

In this section we have presented our experimental results. We ran those experiments for about 40 minutes and averaged them over five iterations for both Series A and B.

[3]git://github.com/roboshepherd/CentralizedTaskServer.git, *hash:* 09187e28292d1cdc179c

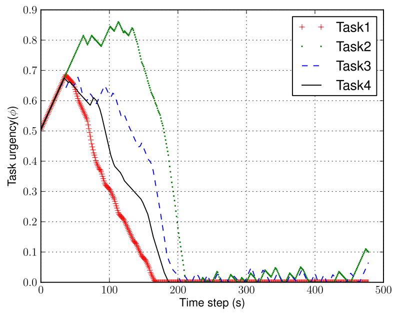


Figure 4.7: Changes in task-urgencies in Series A experiments

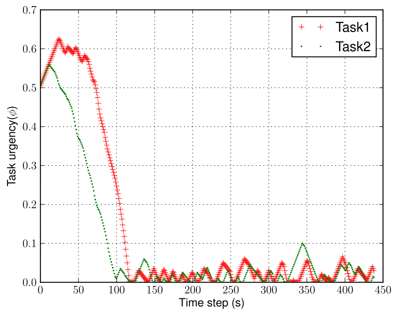


Figure 4.8: Changes in task-urgencies in Series B experiments

Shop-floor work-load history

In our experiments we have defined shop-floor work-load in terms of task-urgencies. For example, Eq. 4.8 shows how we have calculated initial production work-load of our manufacturing shop-floor scenario. Fig. 4.7 and Fig. 4.8 show the dynamic changes in task-urgencies for the single iteration of Series A and Series B experiments respectively. The fluctuations in these plots are resulted from the different levels of task-performance of our robots. In order to measure the task-related work-loads on our system we have summed up the changes in all task-urgencies over time. We call this as shop-floor work-load history and formalized as follows. Let $\phi_{j,q}$ be the urgency of a task j at q^{th} step and $\phi_{j,q+1}$ be the task urgency of $(q+1)^{th}$ step. We can calculate the sum of changes in urgencies of all tasks at $(q+1)^{th}$ step:

$$\Delta\Phi_{j,q+1} = \sum_{j=1}^{M}(\phi_{j,q+1} - \phi_{j,q}) \tag{4.19}$$

From Fig. 4.9 and Fig. 4.10 shows the dynamic shop-floor workload for Series A and Series B experiments respectively. From these plots, we can see that initially the sum of changes of task urgencies (shop-floor workload) is going towards negative direction. This implies that tasks are being served by a high number of robots.

Ratio of active workers

From both Fig. **??** and Fig. **??**, we can see that in production stage, when workload is high, many robots are active in tasks and this ratio varies according to the

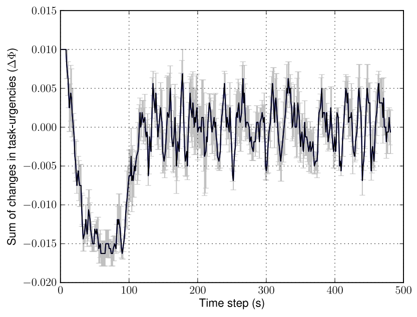


Figure 4.9: Shop-floor workload change history in Series A

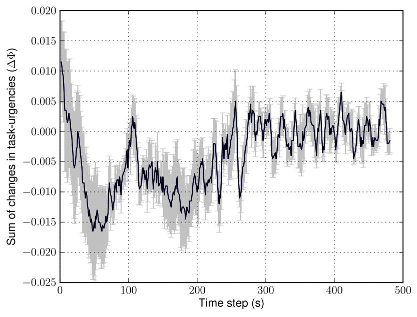


Figure 4.10: Shop-floor workload change history in Series B

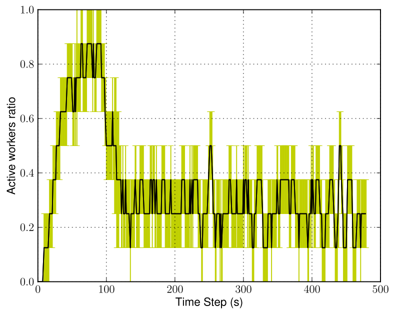


Figure 4.11: Self-organized allocation of robots in Series A

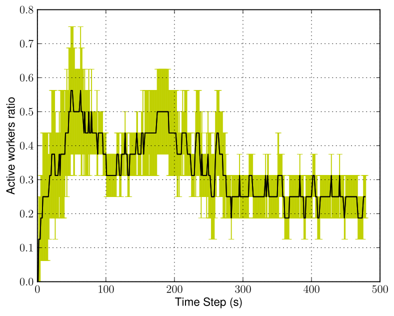


Figure 4.12: Self-organized allocation of robots in Series B

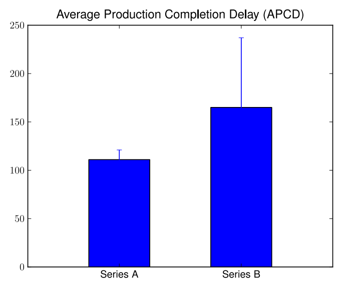


Figure 4.13: APCD of Series A and Series B experiments.

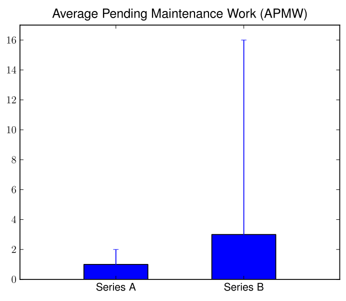


Figure 4.14: APMW of Series A and Series B experiments.

shop-floor work-load changes.

Shop-task performance

In our manufacturing shop-floor scenario, we have calculated the APCD and APMW for both Series A and Series B experiments. For Series A we have got average production completion time 111 time-steps (555s) where sample size is (5 x 2) = 10 tasks, SD = 10 time-steps (50s). According to Eq. 4.12, our theoretical minimum production completion time is 50 time-steps (250s) assuming the non-stop task performance of all 8 robots with an initial task urgency of 0.5 for all 2 tasks and task urgency decrease rate $\Delta\Phi_{DEC}$ = 0.0025 per robot per time-step. Hence, Eq. 4.13 gives us APCD, ζ = 1.22 which means that in Series A experiments, it took 1.22 times more time (305s) than the estimated minimum production completion time. For Series B, we have got average production completion time 165 time-steps (825s) where sample size is (5 x 4) = 20 tasks, SD = 72 time-steps (360s). Hence, Eq. 4.13 gives us APCD, ζ = 2.3. Fig. **??** shows the APCD for both Series A and Series B experiments.

For APMW, Series A experiments give us an average time length of 369 time-steps (1845s). In this period we calculated APMW and it is 1 time-step with SD = 1 time-step (5s) and $\Delta\Phi_{INC}$ = 0.005 per task per time-step. This shows a very low APMW (χ = 0.000235) and a very high robustness of the system. For Series B experiments, from the average 315 time-steps (1575s) maintenance activity of our robots per experiment run, we have got APMW, χ = 0.012756 which corresponds to the pending work of 3 time-steps (15s) where SD = 13 time-steps (65s). This tells us the robust task performance of our robots which can return to an abandoned task within a minute or so. Fig. **??** plots the APMW for both Series A and Series B experiments.

Task specializations

We have measured the task-specialization of the robots based-on their peak value of sensitization. This maximum value represents how long a robot has repeatedly been selecting a particular task. Since tasks are homogeneous we have considered the maxim sensitization value of a robot among all tasks during an experiment run. This value is then averaged for all robots using the following equation.

$$K^G_{avg} = \frac{1}{N}\sum_{i=1}^{N}\max_{j=1}^{M}\left(k^i_{j,q}\right) \tag{4.20}$$

If a robot r_i has the peak sensitization value k^i_j on task j ($j \in M$ tasks) at q^{th} time-step, Eq. **??** calculates the average of the peak task-specialization values of all robots for a certain iteration of our experiments. We have also averaged the time-step values (q) taken to reach those peak values for all robots using the following equation.

$$Q^G_{avg} = \frac{1}{N}\sum_{i=1}^{N} q^i_{k=k_{max}} \tag{4.21}$$

In Eq. **??**, $q^i_{k=k_{max}}$ represents the time-step of robot r_i where its sensitization value k reaches the peak k_{max} as discussed above. By averaging this peak time-step values of all robots we can have an overall idea of how many task-execution cycles are spent to reach the maximum task-specialization value K^G_{avg}. Table **??** and Table **??** show the peak sensitization values of Series A and Series B experiments respectively. Based on Eq. **??** and Eq. **??**, we have got the peak task-sensitization K^G_{avg} values: 0.40 (SD=0.08) and 0.23 (SD=0.04), and their respective time-step Q^G_{avg} values: 38 (SD=13) and 12 (SD=5). They are shown in Fig. **??** and Fig. **??**. Here we can see that the robots in Series A had higher chances of task-specialization than that of Series B experiments. Fig. **??** shows us the task specialization of five robots on Task3 in a particular run of Series B experiment. This shows us how some of the robots can specialize (learn) and de-specialize (forget) tasks over time.

Table 4.3: Peak task-sensitization values of all robots in a Series A experiment.

Robot ID	Maximum k	At time-step (q)
1	0.54	63
2	0.46	63
3	0.47	62
4	0.32	13
5	0.27	10
6	0.20	9
7	0.18	3
8	0.15	2

Table 4.4: Peak task-sensitization values of all robots in a Series B experiment.

Robot ID	Maximum k	At time-step (q)
1	0.64	65
5	0.28	13
6	0.09	1
9	0.09	1
12	0.14	11
13	0.31	18
14	0.13	4
15	0.09	1
16	0.18	3
17	0.16	5
19	0.14	3
22	0.18	19
24	0.41	28
31	0.11	2
35	0.34	11

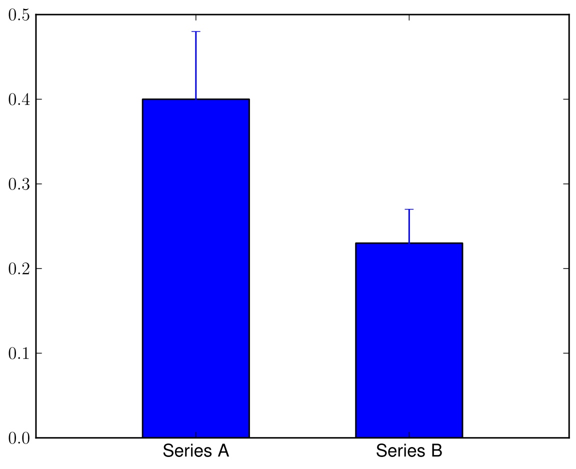


Figure 4.15: Overall task-specialization of robot groups.

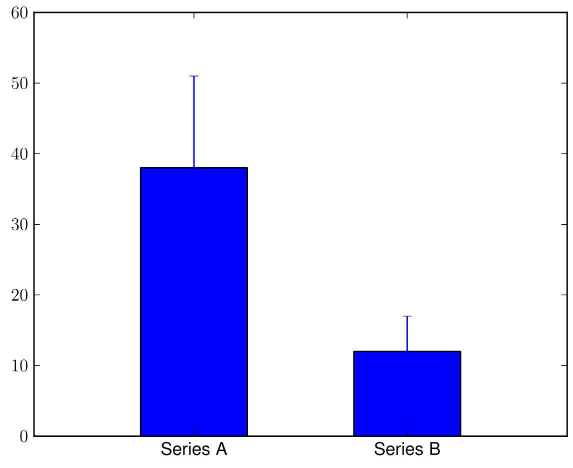


Figure 4.16: Time-steps to reach the peak values of task-specialization

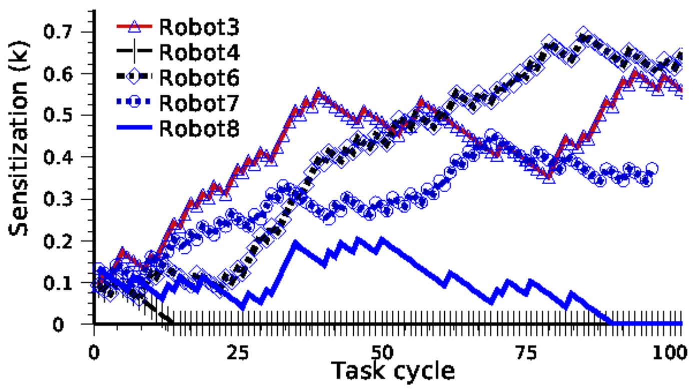


Figure 4.17: Task specialization on Task3 for a Series B experiment.

Robot motions

We have aggregated the changes in translation motion of all robots over time. Let $u_{i,q}$ and $u_{i,q+1}$ be the translations of a robot i in two consecutive steps. If the difference between these two translations be δu_i, we can find the sum of changes of translations of all robots in $(q+1)^{th}$ step using the following equation.

$$\Delta U_{q+1} = \sum_{i=1}^{N} \delta u_{i,q+1} \tag{4.22}$$

The results from Series A and Series B experiments are plotted in Fig. **??** and Fig. **??**. In this plot we can see that robot translations also vary over varying task requirements of tasks.

Communication load

Fig. **??** and Fig. **??** show the number of received TaskInfo signals by each robot in Series A and Series B experiments. Since the duration of each time-step is 50s long and TaskServer emits signal in every 2.5s, there is an average of 20 signals in each time-step.

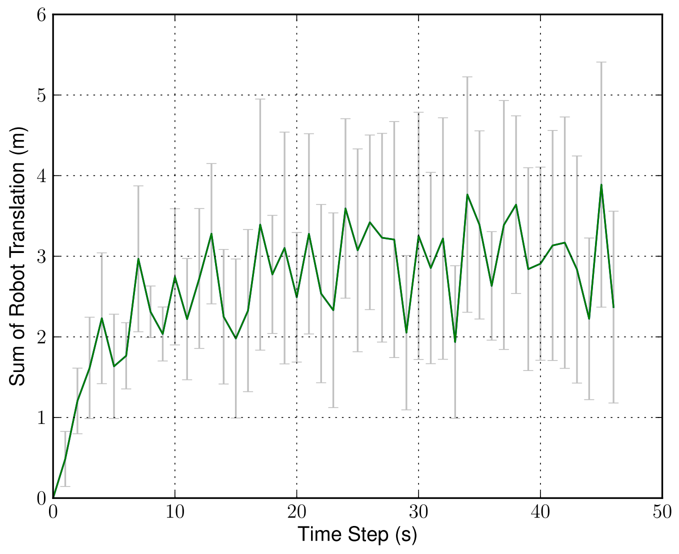


Figure 4.18: Sum of the translations of robots in Series A experiments

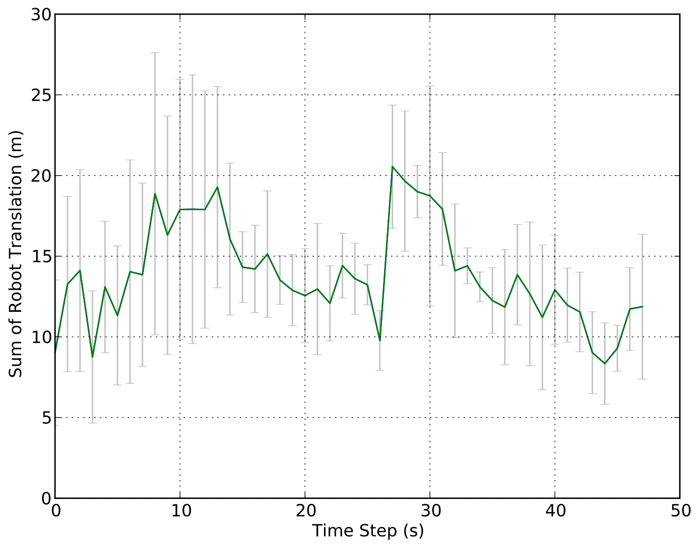


Figure 4.19: Sum of the translations of robots in Series B experiments

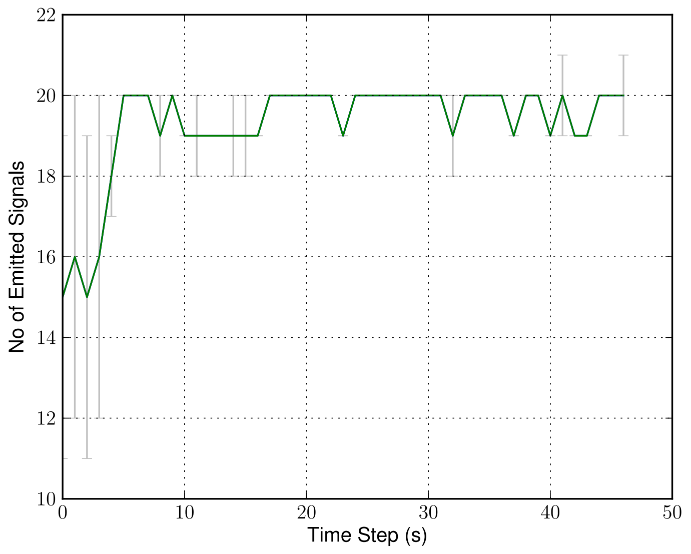


Figure 4.20: Frequency of TaskInfo signalling in Series A experiments

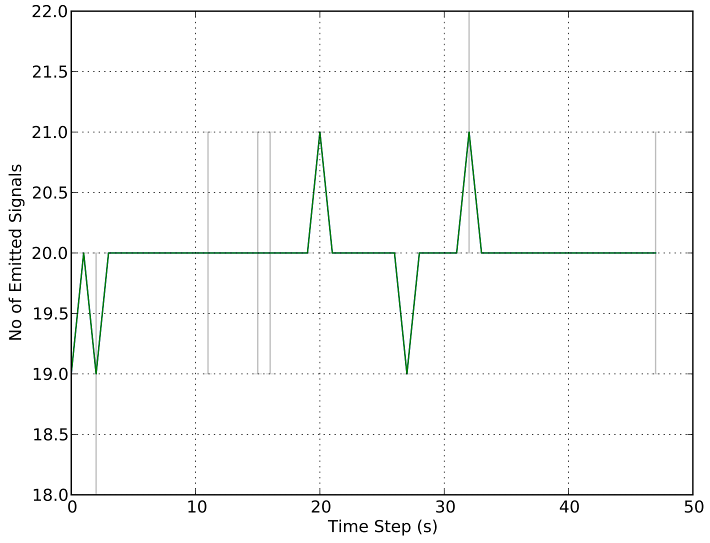


Figure 4.21: Frequency of TaskInfo signalling in Series B experiments

4.7 Discussions

Division of labour

Plasticity

Task-specialization

Flexibility

Concurrency

Robustness

Shop-floor task performance

Information flow

Scaling-up

4.8 Summary and conclusion

In this paper we have validated an inter-disciplinary generic model of self-regulated division of labour (DOL) or or multi-robot task allocation (MRTA) by incorporating it in our multi-robot system (MRS) that has emulated a virtual manufacturing shop-floor activities. A centralized communication system has been instantiated to realize this model. We have evaluated various aspects of this model, such as ability to meet dynamic task demands, individual task specializations, communication loads and flexibility in concurrent task completions. A set of metrics has been proposed to observe the DOL in this system. From our experimental results, we have found that AFM can meet the requirements of dynamic DOL by the virtue of its self-regulatory behaviours. Our centralised communication system broadcasts information to all the robots from a central server. This has the advantage of minimising the communication load and the disadvantage of a single point of failure. In the future, we will explore local peer-to-peer communication models in a MRS having about 40 E-puck robots.

CHAPTER 5

Local Communication for Self-regulated MRTA

As outlined in Sec. 1.2, inspired by the LSLC strategy found in a kind of social wasps, *polybia*, we have intended to carry out a second set of MRTA experiments. Here we have expected that LSLC strategy can give us a necessary system-wide continuous flow of information for achieving the self-regulated DOL by AFM. In this chapter, we have discussed how we have designed our local P2P communication model (LPCM) for MRS, based-on biological local communication strategies. We have presented our implementation algorithms. In order to fit LPCM into our MRS implementation, as explained in Chapter 4, we have added certain functionalities to our previous implementation. In this chapter, we have discussed those additional features. Finally, we have compared the results from both GSNC and LSLC strategies in achieving self-regulated MRTA.

5.1 Motivations

From our understanding of different kinds of communication strategies of biological social systems (Sec. 2.2), this is obvious that an effective LSLC strategy can greatly enhance the self-regulated MRTA. In fact, most swarm robotic researchers now acknowledge that LSLC is the one of the most critical components of a SRS, as global behaviours emerges from local interactions among the individuals and their environment. But how can we design and implement a LSLC scheme among

robots ?
In this study, we have used the concepts of pheromone active-space of ants to realize our simple LSLC scheme. As discussed in Sec. 2.2, ants use various chemical pheromones with different active spaces (or communication ranges) to communicate different messages with their group members. Ants sitting near the source of this pheromone sense and respond quicker than others who wander in far distance. Thus both communication and sensing occurs within a small communication range[1]. We have used this concept of communication range or locality in our LPCM.
In order to find a suitable range (or radius) of communication and sensing, robotic researchers proposed various models that we have briefly discussed in Sec. 2.5.5. Some researchers have stated that this range can be set at design time based on the capabilities of robots. Some other researchers have argued that they can be dynamically varied over time depending on the cost of communication and sensing, e.g. density of peers, ambient noise in the communication channels, or even by aiming for maximizing information spread. In this study, we have followed the former approach as we have assumed that typically robots will not have the precise hardware to dynamically vary their communication and sensing ranges. We have left this issue of selecting best communication range as a topic of future research.

5.2 A locality-based peer-to-peer communication model (LPCM)

5.2.1 General characteristics

Our LPCM relies on the local P2P communications among robots. Robots can communicate to its nearby peers within a certain communication radius, r_{comm}. Here by r_{comm}, we assume that within this distance robots can both sense tasks and exchange communication signals reliably without any significant loss of information. A robot R_1 is a *peer* of robot R_2, if spatial distance between R_1 and R_2 is less than its r_{comm}. Similarly, when a robot comes within this r_{comm} of a task, it can sense this the status of this task. Although the communication and sensing

[1]Although, generally communication and sensing are two different issues, however within the context of our self-regulated DOL, we have broadly viewed sensing as the part of communication process, either implicitly via environment, or explicitly via local peers.

range can be different based on robot capabilities, we have considered them same for simplicity of our implementation.

As shown in Fig. **??**, local communication can also give robots similar task information as in centralized communication mode. It shows that it is not necessary for each robot to communicate with every other robot to get information on all tasks. Since robots can random walk and explore the environment we assume that for a reasonably high robot-to-space density, all task will be known to all robots after an initial exploration period. In order to update the urgency of a task, robots can estimate the number of robots working on a task in many ways: such as, by using their sensory perception (e.g., camera), by doing local P2P communication with others. In Fig. **??** we have shown that robots exchange both task information and self status signals to peers.

We characterize our communication model in terms of three fundamental issues: 1) message content (*what to communicate*) 2) communication frequency (*when to communicate*) and 3) target recipients (*with whom to communicate*) (**?**). In a typical MRS, message content can be categorized into two types: 1) state of each individual robot and 2) target task (goal) information (**?**). The latter can also be subdivided into two types: 1) an individual robot's target task information and 2) information of all available tasks found in the system.

Regarding the first issue, our communication model is open. Robots can communicate with their peers with any kind of message. Our model addresses the last two issues very specifically. Robots communicate only when they meet their peers within a certain communication radius (r_{comm}). Although in case of an environment where robots move relatively faster the peer relationships can also be changed dynamically. But this can be manipulated by setting the signal frequency and robot to space density to somewhat reasonably higher value.

In terms of target recipients, our model differs from a traditional publish/subscribe communication (PSC) model by introducing the concept of dynamic subscription. In a traditional PSC model, subscription of messages happens prior to the actual message transmission. In that case prior knowledge about the subjects of a system is necessary. But in our model this is not necessary as long as all robots uses a common addressing convention for naming their incoming signal channels. In this way, when a robot meets with another robot it can infer the address of this peer robot's channel name by using a shared rule. A robot is thus always listening to its own channel for receiving messages from its potential peers or message publishers. On the other side, upon recognizing a peer a robot sends a message to this particular peer. So here neither it is necessary to create any custom subject namespace (e.g., as in **?**) nor we need to hard-code information in each robot controller about the knowledge of their potential peers *a priori*. Subscription is done automatically based on their respective r_{comm}.

5.2.2 Implementation algorithm

Within the context of our self-regulated MRTA experiments under AFM, here we have formalized these aspects of LPCM into an implementation algorithm. It has three major aspects:

1. local sensing of peers and tasks,
2. listening to the task-information peer signals and
3. emitting local task-information signals to peers. Here we present a typical implementation.

Algorithm 1: Locality based P2P Communication Model

```
1:  Input: $RobotID, r_{comm}, r_{task}, TaskInfoDB, RobotPose,$
2:         $ListeningBuffer, EmissionBuffer$
3:  Output: updated $TaskInfoDB$
4:  comment: Perception of a task, if the robot is within $r_{task}$
5:  $TaskPose \leftarrow$ Estimate/get pose of a task within $r_{task}$
6:  $TaskUrgency \leftarrow$ Estimate/get urgency of a task within $r_{task}$
7:  $TaskInfo \leftarrow (TaskPose, TaskUrgency)$
8:  $TaskInfoDB \leftarrow$ UpdateTaskInfoDB($TaskInfo$)
9:  comment: Listening of LocalTaskInfo signal(s) from peers
10: $RobotPeers \leftarrow$ Identify/get a list of peers within $r_{comm}$
11: for all peer $\in RobotPeers$ do
12:     if (caught a LocalTaskInfo signal from nearby peer) then
13:         $ListeningBuffer \leftarrow$ LocalTaskInfo of peer
14:         $TaskInfoDB \leftarrow$ UpdateTaskInfoDB($ListeningBuffer$)
```

```
15:     end if
16: end for
17: comment: Emitting of own LocalTaskInfo signal to peers
18: $EmissionBuffer \leftarrow TaskInfoDB$
19: for all peer $\in RobotPeers$ do
20:     EmitLocalTaskInfoSignal(peer, $EmissionBuffer$ )
21: end for
22: $RobotPeers \leftarrow 0$
```

From Algorithm 1, we see that a robot controller is initialized with its specific $RobotID$ and default values of r_{comm} and r_{task} that correspond to the robot's communication and sensing range respectively. We have assumed that these values are same for all robots and for all tasks. Initially a robot has no information about tasks, i.e. $TaskInfoDB$ is empty. It has neither received nor transmitted any information yet, i.e. corresponding data buffers, $ListeningBuffer$ and $EmissionBuffer$, are empty. Upon sensing a task, robot determines the task's pose and urgency of that task (according AFM rules described in Sec. 4.2.5). This is not strictly necessary as this information can also be available from alternate sources, e.g. via communicating with a TPS.
Robot controller then executes the **UpdateTaskInfoDB()** that modified the robot's information about the corresponding task. In second step, robot senses its nearby peers located within r_{comm} and add them in $RobotPeers$. The potential LocalTaskInfo signal reception from a peer again triggers the **UpdateTaskInfoDB()** function. In the last step, robot emits its own task information as a LocalTaskInfo signal to its peers. Finally it cleans up the current values of $RobotPeers$ for running a new cycle.

5.3 Experiment Design

Our local communication experiments follow the similar design of experimental parameters and observables of Series B experiments as outlined in Sec. 4.4. Table 5.1 highlights the major parameters of these experiments. Here we can see that there are two new parameters in Series C and Series D experiments, i.e. communication and sensing ranges. For the sake of simplicity, both of these values are kept

Table 5.1: Two series of local communication experiments

Parameter	Series C | Series D
task-perception range (r_{task})	$0.5m \mid 1m$
Communication range (r_{comm})	$0.5m \mid 1m$
Total number of robots (N)	16
Total number of tasks (M)	4
Experiment area (A)	4 m^2

same in both series of experiments.

Series D uses larger ranges that enables our robot to capture more task information at a time. That is similar to a working “global sensing and global communication” of information by the robots, since in a large group of robots, it is not practical that a robot communicate with all other robots. On the other hand, Series C uses the half of the values of Series D. This enables our robots to capture less information, that mimics a true LSLC strategy. The main motivation for using two different ranges is to find any significant difference in performance from these two types of experiments. Although these differences may not be significant in many times, they can reflect the rationale behind our selection of ranges without being purely arbitrary.

5.4 Implementation

On top of the base implementation of our MRS, we have extended our MRS so that we can fit our LPCM into it. The additional communication interfaces and changes in RCC in TPS implementation are mentioned here. The details of our MRS implementation can be found in Sec. 4.5.

Additional communication interfaces under LPCM

We have added three new D-Bus signal interfaces in this version of our MRS. They are briefly discussed below.

RobotPeers: This signal is emitted by SwisTrack to each robot’s signal path in a common “ac.uk.newport.ril.SwisTrack” interface. The payload of this signal contains the list of IDs of peers of each robot within it’s r_{comm}. These peer-IDs are

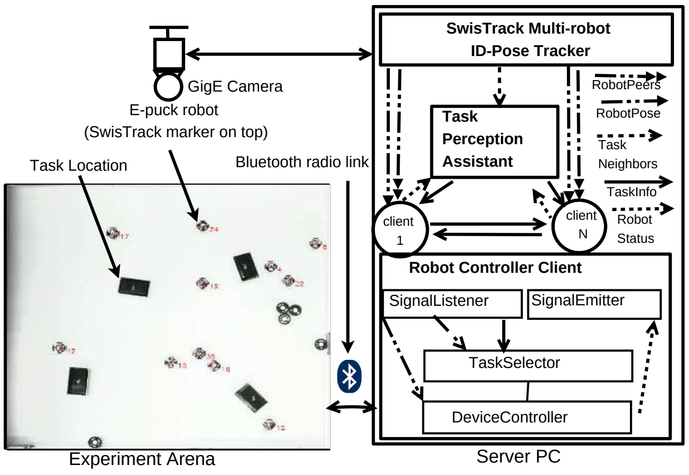


Figure 5.1: Hardware and software setup under local communication experiments.

extracted by analysing the captured image frame. This signal is caught by RCCs and used in emitting the LocalTaskInfo signal (discussed below).

TaskNeighbohood: This signal is emitted by SwisTrack for TPS and the payload of this signal contains the list of IDs of robots that are co-located within a task's perception range, r_{task}. TPS catches one signal per task and based-on the robot IDs, it emits the TaskInfo signal to those robot's RCC path (as discussed in Sec. 4.5).

LocalTaskInfo: This signal is emitted by RCCs to their peers that are co-located within the same communication range r_{comm}. The payload of this signal contains the known task information of a RCC. This task-information can be gathered either through task-perception (via TaskInfo signal received from TPS) or via communications i.e. through LocalTaskInfo signals from its peers.

Modifications in RCC and TPS

As shown in Fig. 5.1, RCCs disseminate task information to each other by *LocalTaskInfo* D-Bus signal. The contents of task information are physical locations of tasks and their urgencies. However as our robots are incapable of sensing task directly. So it relies on TPS for a task-perception signal, TaskInfo, that contains the actual task location and urgency. When a robot r_i comes within r_{task} of a task j, SwisTrack reports it to TPS (by *TaskNeighbors* signal) and TPS then gives it current task information to r_i by *TaskInfo* signal. As we have discussed, TPS catches feedback signals from robots via *RobotStatus* signal. This can be used to inform TPS about a robot's current task id, its device status and so on. TPS uses this information to update relevant part of task information such as, task-urgency. This up-to-date information is encoded in next TaskInfo signal.

From the above discussions, we can see that our base implementation of RCC's task-allocation (i.e. TaskSelector) is almost unchanged in this local implementation. The only thing we need to extend is the D-Bus signal reception and emission interfaces that catches the above signals and put the signal payload in DataManager. For the sake of simplicity, we have merges the perceived TaskInfo with communicated LocalTaskInfo into one so that TaskAllocator always gets all available task information. We have not made any major change in other modules of RCC, i.e. DeviceController. The full Python implementation of this RCC is available for download from the author's GitHub repository[2].

In our base implementation TPS periodically broadcasts TaskInfo signals to all robots. But in this local version, TPS only emits TaskInfo signal when a robot is within a task's perception range, r_{task} based-on the TaskNeighbour signal from SwisTrack. For this additional interface, we have extended D-Bus SignalListener to catch this additional D-Bus signal. No major changes ared done in the other part of TPS implementation. The full Python implementation of TPS is available for download from this GitHub repository[3].

5.5 Results and discussions

In this section we have presented our experimental results. We ran those experiments for about 40 minutes and averaged them from three iterations. For comparison purposes, here we present some of the results of our baseline experiments in centralized communication mode. Details can be found in (**?**).

Fig. 5.2 shows the dynamic changes in task urgencies. In order to describe our system's dynamic behaviour holistically we analyse the changes in task urgencies over time. Let $\phi_{j,q}$ be the urgency of a task j at q^{th} step. In $(q+1)^{th}$ step, we can find the change of urgency of task j :

[2]http://github.com/roboshepherd/EpuckDistributedClient
[3]http://github.com/roboshepherd/DitributedTaskServer

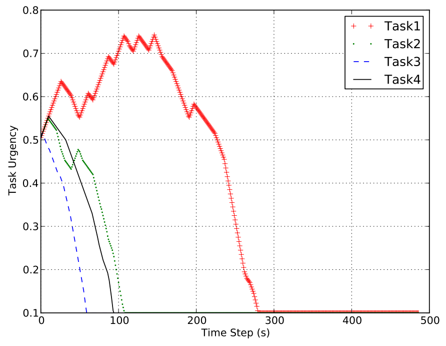


Figure 5.2: Task urgencies observed at TaskServer in local mode r_{comm}=0.5m

$$\delta\phi_{j,q+1} = (\phi_{j,q+1} - \phi_{j,q}) \tag{5.1}$$

So we can calculate the sum of changes in urgencies of all tasks at $(q+1)^{th}$ step:

$$\Delta\Phi_{j,q+1} = \sum_{j=1}^{M} \delta\phi_{j,q+1} \tag{5.2}$$

Fig. 5.3 plots this sum of changes of task urgencies by a dashed line. If we consider the absolute change over a window of time w in the following equation, we can describe the overall changes of our systems in both positive and negative directions.

$$\Delta\Phi_{jw,q+1} = \sum_{j=0}^{w-1} \mid \Delta\Phi_{q+j} \mid \tag{5.3}$$

In order to find convergence in MRTA we have calculated the sum of absolute changes in task urgencies over a window of 2 consecutive steps (100s). This is plotted in solid line in Fig. 5.3. Note that we scale down the time steps of this plot by aggregating the values of 10 consecutive steps (50s) of Fig. 5.2 into a single step value. From Fig. 5.3 we can see that initially the sum of changes of task urgencies are towards negative direction. This implies that tasks are being served by a high number of robots. When the task urgencies stabilize near zero the fluctuations in urgencies become minimum. Since robots chose tasks stochastically, there will always be a small changes in task urgencies. A potential convergence point is located by considering the persistence existence of the value of $\Delta\Phi_{jw,q+1}$ below a threshold 0.1. Tn Fig. 5.3 this convergence happens near step 23 or after 1150s from the beginning of our experiments. This implies that from this point of time

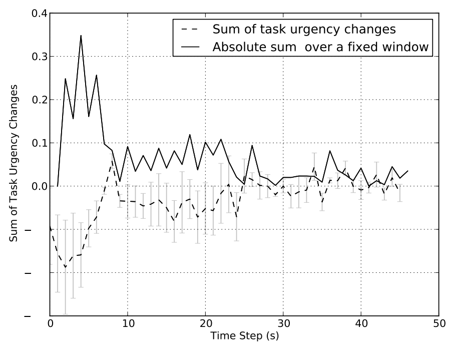


Figure 5.3: Convergence of task urgencies in centralized mode

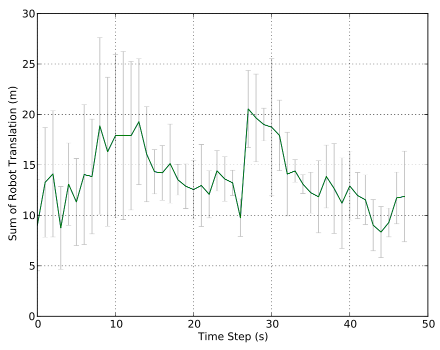


Figure 5.4: Sum of translations of all robots in centralized mode

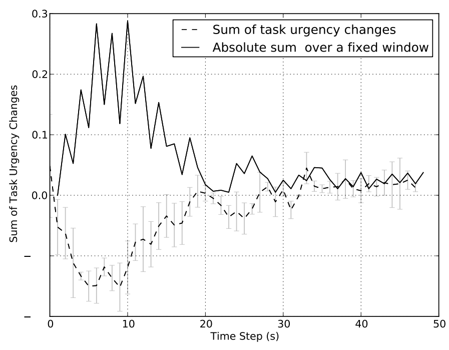


Figure 5.5: Convergence of task urgencies in local mode r_{comm}=0.5m

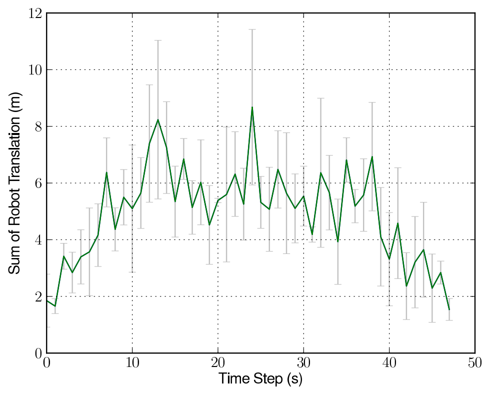


Figure 5.6: Sum of translations of all robots in local mode r_{comm}=0.5m

and onwards, changes of our system's behaviour remain under a small threshold value.

Using this same criteria, we can find that in local communication experiment convergence happens after step 14 in case of r_{comm}=0.5m (Fig. 5.5) and after step 29 in case of r_{comm}=1m (Fig. 5.7).

We have aggregated the changes in translation motion of all robots over time. Let $u_{i,q}$ and $u_{i,q+1}$ be the translations of a robot i in two consecutive steps. If the difference between these two translations be δu_i, we can find the sum of changes of translations of all robots in $(q+1)^{th}$ step using the following equation.

$$\Delta U_{q+1} = \sum_{i=1}^{N} \delta u_{i,q+1} \tag{5.4}$$

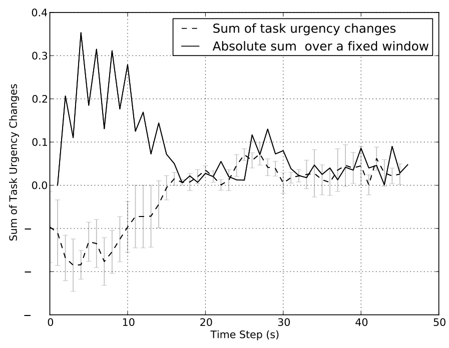


Figure 5.7: Convergence of task urgencies in local mode r_{comm}=1m

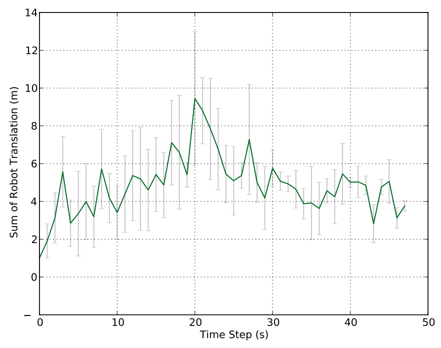


Figure 5.8: Sum of translations of all robots in local mode r_{comm}=1m

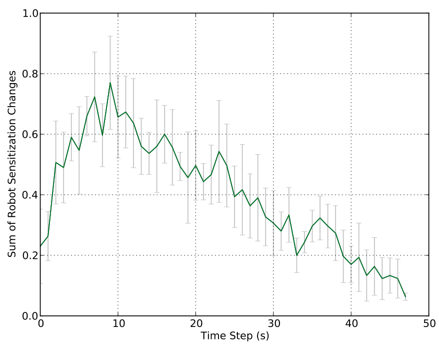


Figure 5.9: Changes in sensitizations in local mode r_{comm}=0.5m

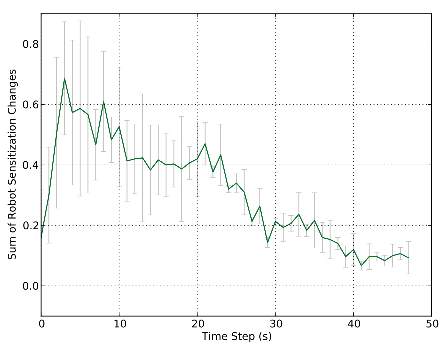


Figure 5.10: Changes in sensitizations in local mode r_{comm}=1m

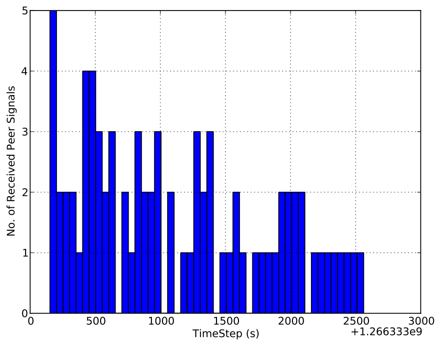


Figure 5.11: Peer signals caught by Robot12 in local mode r_{comm}=0.5m

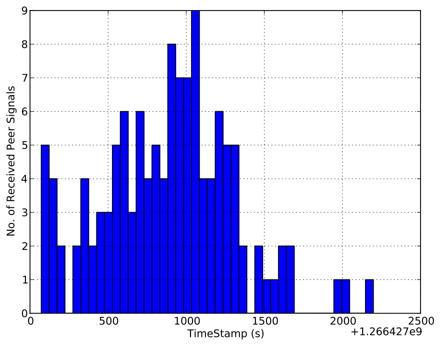


Figure 5.12: Peer signals caught by Robot12 in local mode r_{comm}=1m

The result from centralized communication experiment is plotted in Fig. 5.4. In this plot we can see that robot translations also vary over varying task requirements of tasks. But it fails to show a consistence behaviour like previous plots. The robot translation results from local mode experiments are plotted in Fig. 5.6 and Fig. 5.8. The reduction of robot translation is significant in both local communication experiments.

Similar to Eq. 5.2, we can calculate the absolute sum of changes in sensitizations by all robots in the following equation.

$$\Delta K_{j,q+1} = \sum_{j=1}^{M} \mid \Delta k_{j,q+1} \mid \tag{5.5}$$

This values of ΔK from local experiments are plotted in Fig. 5.9 and Fig. 5.10. They show that the overall rate of learning and forgetting decrease over time. It is a consequence of the gradually increased task specializations of robots.

As an example of P2P signal reception of a robot, Fig. 5.11 and Fig. 5.12 show the number of received signals by Robot12 in two local experiments. It states the relative difference of peers over time in two local cases. The overall P2P task information signals of both of these local modes are plotted in Fig. 5.13 and Fig. 5.14. Note that in centralized communication mode the task info signal frequency was kept 20 signals per time step (**?**).

As an example of task specialization of a robot we plotted sensitization of Robot12 in Fig. 5.15. It shows that this robot has specialized in Task1. The continuous learning of Task1 happens from step 23 to step 59 where it has learned this task

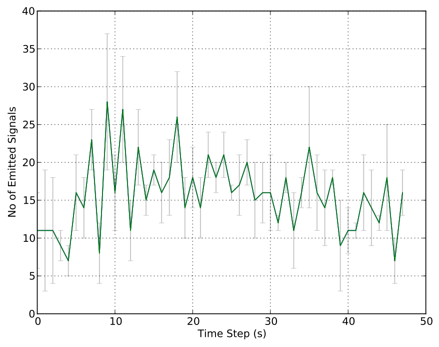


Figure 5.13: Frequency of P2P signalling in local mode r_{comm}=0.5m

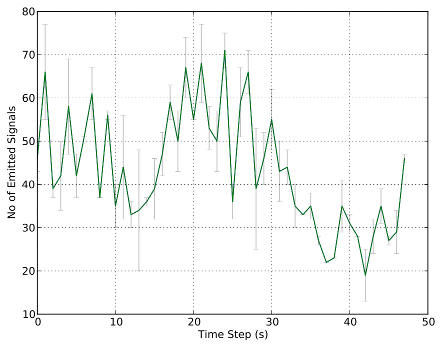


Figure 5.14: Frequency of P2P signalling in local mode r_{comm}=1m

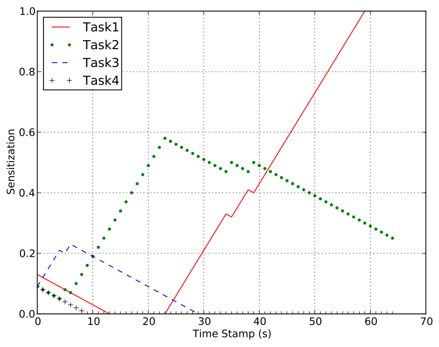


Figure 5.15: Task specialization of Robot12 in local mode r_{comm}=0.5m

Figure 5.16: Changes in translation of Robot12 in local mode r_{comm}=0.5m

completely. This behaviour was found common in all robots with varying level of sensitizations. Hence we get the linear decrease of ΔK in sensitization plots. However, the changes in motion of this robot plotted in Fig. 5.16 was not stable due to the fact that robots frequently avoid dynamic obstacles and select random-walking. Krieger and Billeter presented MRTA in a team of 12 robots in ants' foraging scenario where a central server sent broadcast message containing energy level of the colony to all robots (**?**). They also applied a form of indirect communication in order to share information about discovered food sources among robots. Although this P2P communication did not happen in real time, it improved the robustness of peer recruitment when compared with centralized communication only. Agassounon and Martinoli compared three task-allocation algorithms using simulations and reported that algorithm that used information sharing among local peers became robust in worker allocation (**?**). However, environment condition should be known *a priori* to optimize some parameters. By modulating the transmission power of local on-board wireless device of E-puck robots, Cianci *et al.* enabled them to communicate in different local radii (**?**). In a self-organized decision making experiment, decision of 15 robots converged faster when local communication radius was relatively higher. Since the number of tasks was only two (left or right wall following), robots got global view of the system with increased communication radius. Thus they agreed quickly in one task.

5.6 Summary and conclusion

We presented a locality based dynamic P2P communication model that achieves similar or better self-regulated multi-robot task allocation (MRTA) than its centralized counterpart. Particularly the reduction in robot movement states that local model is preferable when we need to minimize robot energy usage. Our model assumes no prior knowledge of the environment and it also does not depend on the number of the robots. We presented an abstract algorithm that can be used with various P2P communication technologies. We reported our implementation of this algorithm by using D-Bus technology in Linux. Comparative results from local mode experiments shows us that robots relatively perform better when the radius of communication is such small that most of the robots only communicate with their closest peers. This is contrary to the findings from (**?**, **?**) where increased amount of information help to get desired task-allocation quickly. In our case, more information exchange with larger communication radius or with centralized communication increased task switching among robots. In case of local information exchange, robots have more chances to select local tasks by the virtue of self-regulating principles. Thus in local communication mode our system converged better and significantly reduced robot motions.

Our future work include performing more experiments with increased number of robots (up to 40) and tasks. Since our existing Bluetooth communication setup with one server fail to scale we are looking forward to distribute communication loads over a network cluster of tens of servers.

CHAPTER 6

Conclusions

6.1 Summary of contributions

6.2 Future work