Self-regulated Multi-robot Task Allocation: A Taxonomy and Comparison of Centralized and Local Communication Strategies

Md Omar Faruque Sarker and Torbjørn S. Dahl[a]

[a]*Cognitive Robotics Research Centre*
Newport Business School, Allt-yr-yn Campus
Newport, NP20 5DA, United Kingdom.
Mdomarfaruque.Sarker |Torbjorn.Dahl@newport.ac.uk

Abstract

To deploy a large group of autonomous robots in dynamic multi-tasking environments, a suitable multi-robot task-allocation (MRTA) solution is required. This paper proposes to solve the MRTA problem using a set of previously published generic rules for division of labour derived from the observation of ant, human and robotic social systems. The concrete form of these rules, the *attractive filed model* (AFM), provides sufficient abstraction to local communication and sensing which is uncommon in existing MRTA solutions. We have validated the effectiveness of AFM to address MRTA using two bio-inspired communication and sensing strategies: "global sensing - no communication" and "local sensing - local communication". The former is realized using a centralized communication system and the latter is emulated under a peer-to-peer local communication scheme. They are applied in a manufacturing shop-floor scenario using 16 e-puck robots. A flexible multi-robot control architecture, *hybrid event-driven architecture on D-Bus*, has been outlined which uses the state-of-the-art D-Bus interprocess communication. Based-on the organization of task-allocation, communication and interaction among robots, a novel taxonomy of MRTA solutions has been proposed to remove the ambiguities found in existing MRTA solutions. Besides, a set of domain-independent metrics, e.g., plasticity, task-specialization and energy usage, has been formalized to compare the performances of the above two strategies.

1. Introduction

Multi-robot systems can provide improved performance, fault-tolerance and robustness in complex and distributed tasks through parallelism and redundancy [1, 2]. However in order to get potential benefits of multi-robot systems, we need to answer a common research question. *How can we allocate tasks among multiple robots dynamically?* Traditionally, this issue has been identified as the *multi-robot task allocation* (MRTA) [3]. This issue can be treated as the *division of labour* (DOL) among robots, analogous to the DOL in biological and human social systems[1] [4].

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 [5]. This is a *NP-hard* optimal assignment problem where optimum solutions can not be found quickly for large and complex problems [6].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 multi-robot system, 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.

Early research on predefined task-allocation was dominated by intentional coordination [6], use of dynamic role assignment [7] and market-based bidding approach [8]. Under these approaches, robots use direct task-allocation method, often to communicate with group members for negotiating on tasks. These approaches are intuitive, comparatively straight forward to design and implement and can be analysed formally. However, these approaches typically works well only when the number of robots are small (≤ 10) [9].

On the other hand, self-organized task-allocation approach relies on the emergent group behaviours, such as emergent cooperation [10], adaptation rules [11] 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 suboptimal. 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 [12]. Our approach is inspired from the studies of emergence of task-allocation in both biological and human social 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 later. Primarily these rules

[1] Although the term "division of labour" is often used in biological literature and the term "task-allocation" is primarily used in multi-agent literature, in this paper we have used these terms interchangeably.

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 to employ these rules in the individual level, we have developed a formal model of self-regulated DOL, called the *attractive field model* (AFM).

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 rely heavily 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 provides a rich abstraction to this requirement through a system-wide continuous flow of information about tasks, agent states etc. This flow of information can be achieved by using both centralized and decentralized communication modes under explicit and implicit communication strategies.

In order to enable continuous flow of information in our multi-robot system, we have implemented two types of sensing and communication strategies inspired by the self-regulated DOL found in two types of social wasps: *polistes* and *polybia* [13]. Depending on the group size, these species follow different strategies for communication and sensing of tasks. Polistes wasps are called the *independent founders* 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* 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: independent founders do not rely on any cooperative task performance while swarm founders interact with each-other locally to accomplish their tasks. The work mode of independent founders 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 swarm founders 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.

The main contributions of this paper are as follows:

- Interpretation of 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.

2. Related work

Since 90s, MRTA many robot control architectures have been solely designed to address MRTA issue from different perspectives. Based-on the high-level design of those solutions, here we have classified them into two major categories: 1) predefined or intentional task-allocation and 2) bio-inspired self-organized task-allocation. Fig. 1 illustrates our classification.

In most of the traditional multi-robot system, task allocation is done using 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. Knowledge-based and multi-agent based approaches use various knowledge-based techniques to represent tasks, robot capabilities etc. One of the early well-known MRTA architecture of this category was 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 [14]. Robots use these models to select a task that benefit the group as a whole.

Similar to ALLIANCE, multi-agent based task allocation also use both centralized and decentralized approaches for allocating tasks among its peers. [15] 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.

As a feasible alternative to the above common multi-agent based task-allocation techniques, many researchers have been following the market-based bidding approach [8]. Originated from the 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 perform a 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 perform 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 by the manager with a mutual agreement with contractor that maximizes the individual profits.

Under role or value-based task-allocation scheme, each role assumes several specific tasks and each robot selects roles that best suit their individual skills and capabilities [7]. In this case, robots are typically heterogeneous, each one having variety of

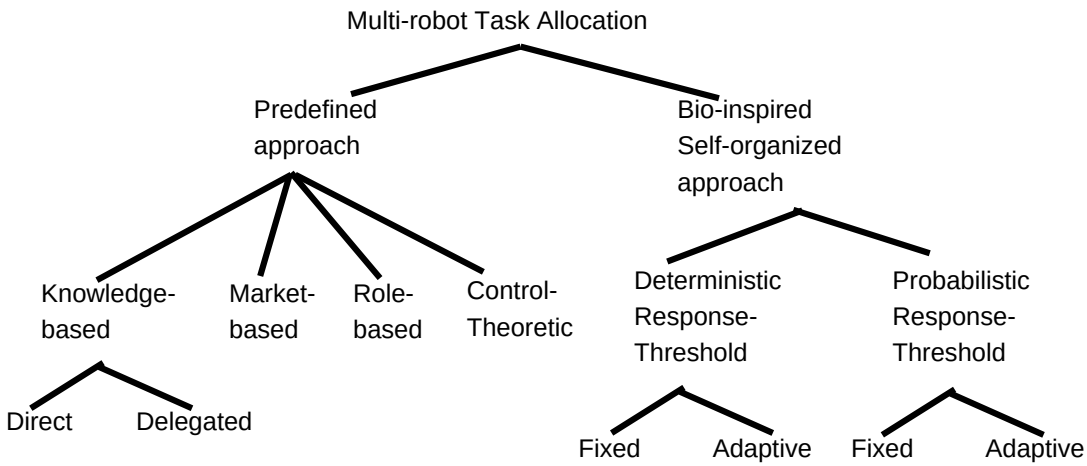


Figure 1: Classification of MRTA solutions.

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 [16], positions played by different robots are often defined as roles, e.g. goal-keeper, left/right defender, left/right forwarder etc.

Under control-theoretic approaches, 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 [17], multi-robot box-pushing [18] etc.

Predefined task-allocation through few other approaches are also present in the literature. For example, inspired by the vacancy chain phenomena in nature, [19] proposed a vacancy chain scheduling algorithm for a restricted class of MRTA problems in spatially classifiable domains.

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 using the principles of self-organization, e.g. positive and negative feedback mechanisms, randomness. Moreover interaction among individuals and their environment are modulated by the stigmergic, local and broadcast communications. Among many variants of self-organized task-allocation, most common type is threshold-based task-allocation [20]. In this approach, a robot's decision to select a particular task depends largely on its perception of stimulus (demand for a task) and its corresponding response threshold for that task.

Under deterministic response-threshold 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 this stimuli. When the task-stimuli falls below the fixed threshold the robot abandons that task. This type of approach has been effectively applied in foraging [21, 11], aggregation [22]. This fixed response-threshold can initially be same for all robots [23], or they can be different according robot capabilities or configuration of the system [21]. 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 [20, 22].

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. Robots always have small nonzero probabilities for all tasks.

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

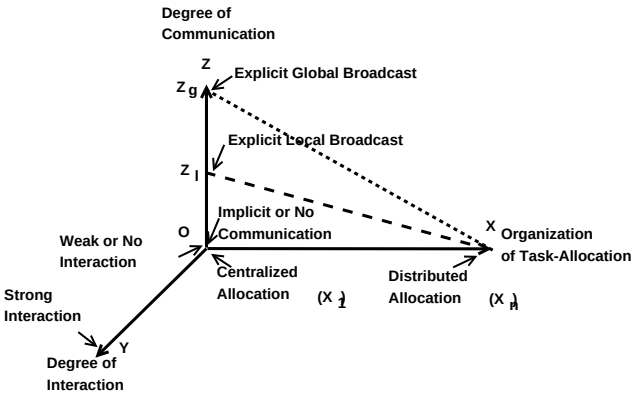


Figure 2: Three major axes of complexities in MRTA

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 of robots and dynamic tasks. In an arbitrary event handling domain, [24] 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.

[5] presented a comparative study of the complexity and optimality of key architectures, e.g. ALLIANCE [14], BLE [25], M+ [26], MURDOCH [27], First piece auctions [28] and Dynamic role assignment [7], all of them relied upon predefined task-allocation methods. The computational and communication requirements of these MRTA solutions 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.

From above discussions we can see that, 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. So for large multi-robot system, self-organized task-allocation methods can potentially be selected, if the complex tasks can be divided into simple pieces that can be carried out by multiple simple robots in parallel with limited communication and interaction requirements.

3. A three-axes taxonomy of MRTA solutions

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 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 Fig. 2, 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 one of the robots (aka leader) to manage the task-allocation. 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, a small number of robots can have fully distributed task-allocation where each robot acts as an independent task-allocator (e.g. as discussed before 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 [29] show that a swarm can be guided to a target by a few informed individuals (or leaders) while maintaining the self-organizing principles of 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.

In Fig. 2, Y axis corresponds to the level of robot-robot interaction present in the system. Group-level interaction can be classified into various levels: collective, cooperative, coordinative and collaborative [6]. The presence of interaction can be due to the nature of the problem, e.g. 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 multi-robot systems 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 with high degrees of interactions among the robots suffer from the performance loss if some of the robots removed from the system.

The Z axis of Fig. 2 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 a 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 the 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 solutions.

The central issue of this paper is to determine the role of communication and sensing strategies under an adaptive response-threshold task-allocation method. So we have focused to examine the benefits of traversing along the various axes of Fig. 2. In this paper, 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).

4. Robotic interpretation of the Attractive Field Model

The construction of AFM has been achieved through a series of collaborative interactions among our project partners. From the biological experiments of ants colonies *Temnothorax albipennis* we inferred the bottom-up 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 studies helped us to formalize the generic rules into a concrete model and to validate this model by deploying it in our robot controllers within the context of a manufacturing shop-floor scenario. In this section, we have described the robotic validation of AFM, with a brief presentation on interpretations of AFM from different social perspectives.

4.1. Generic framework

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

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

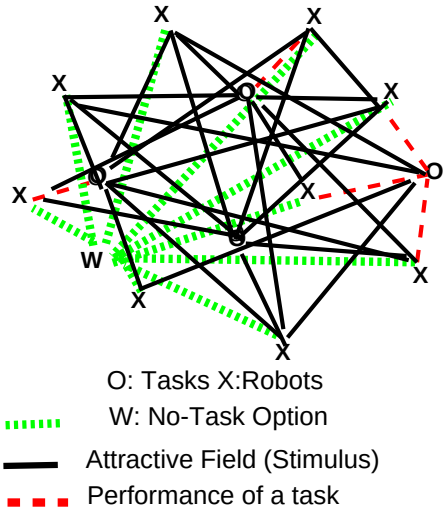


Figure 3: The attractive filed model (AFM)

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.

Rule 2: Sensitization. Self-regulatory social systems allow the differentiation in the use of (or access to) information, e.g. through sensitization or learning of 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 social 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 social 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-regulatory social systems, 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 primarily depends upon

the distance between the task and the agent. This relationship is represented using weighted links. Besides, 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. 3. 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 becomes different, 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. Moreover, 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 (Fig. 3), the nodes have arbitrary places. 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.

In summary, from the above diagram of 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. Relationship of AFM with self-organization

It is interesting to note that our proposed four generic rules can be considered as the four major foundations of a self-organized system (Fig. 4). Self-organized systems exhibit four distinct perspectives known as so-called ingredients or properties of self-organization [30]. However, it is not clear how those properties can come into existence. Here, we have described the four underlying mechanisms that explains how self-organization can be realized in different social systems using our generic framework of self-regulation. From our understanding, we can explain it in the following ways.

Firstly, multiple interactions become 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 swarm intelligence 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 swarm intelligence, 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 gradual decrease of response-threshold of individuals which increases the 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 [12].

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.

4.3. Interpretation of AFM in multi-robot systems

The interpretation of AFM in a multi-robot system 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.

$$If\ the\ task\ is\ not\ being\ done: \ \phi_j \rightarrow \phi_j \ + \delta_{\phi_{INC}} \quad (1)$$

$$If\ the\ task\ is\ being\ done: \ \phi_j \rightarrow \phi_j \ - n\,\delta_{\phi_{DEC}} \quad (2)$$

Eq. 1 refers to a case where no robot attend to task j and Eq. 2 refers to another case where n robots are concurrently performing the task j.

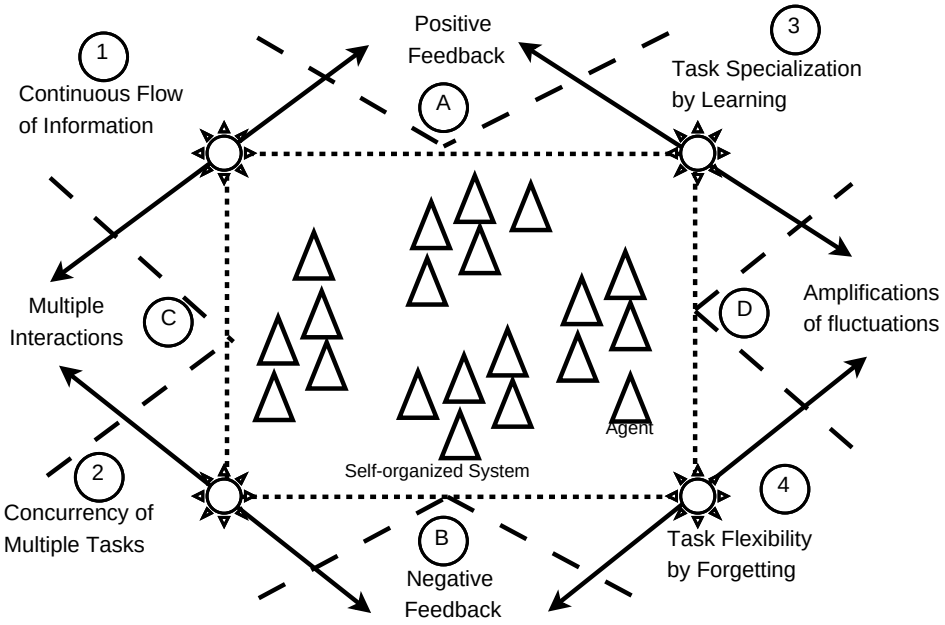


Figure 4: Four generic rules establish the self-regulated DOL in social systems

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^i_j. If a robot i does not do a task j for some time, it forgets about j and k^i_j is decreased, by another small amount, say k_{DEC} . Thus a robot's task-sensitization update follows these rules.

$$If\ task\ is\ done:\ k^i_j \rightarrow k^i_j\ +\ k_{INC} \tag{3}$$

$$If\ task\ is\ not\ done:\ k^i_j \rightarrow k^i_j\ -\ k_{DEC} \tag{4}$$

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 [12], we can formally encode this stimuli of attractive field as follows.

$$S^i_j = tanh\{\frac{k^i_j}{d_{ij}+\delta}\phi_j\} \tag{5}$$

$$S^i_{RW} = tanh\left\{1 - \frac{\sum_{j=1}^{J} S^i_j}{J+1}\right\} \tag{6}$$

$$P^i_j = \frac{S^i_j}{\sum_{j=0}^{J} S^i_j}\quad where,\quad S^i_0 = S^i_{RW} \tag{7}$$

Eq. 5 states that the stimuli of a robot r_i to a particular task j, S^i_j depends on r_i's spatial distance to j (d_{ij}), level of sensitization to j (k^i_j), and perceived urgency of that task (ϕ_j). In Eq. 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^i_j is a probability function, it is chosen as a *tanh* in order to keep the values between 0 and 1. Eq. 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. 7 which states that the probability of choosing a task j by robot r_i is directly proportional to its calculated stimuli S^i_j. 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.

5. AFM based task-allocation solution

We have designed a set of manufacturing shop-floor scenario experiments for validating the effectiveness of our AFM in producing self-regulated 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 our manufacturing shop-floor scenario and our task-allocation solution for robot-controllers.

5.1. A manufacturing shop-floor scenario

By extending our interpretation of AFM in multi-robot system, we can set-up manufacturing shop-floor 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

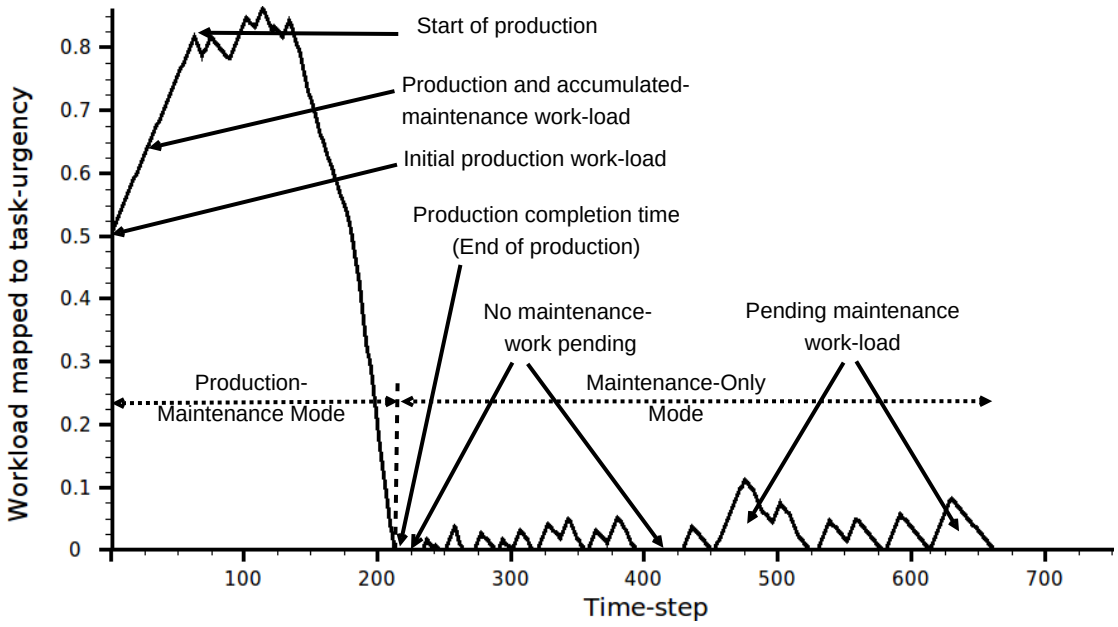


Figure 5: A manufacturing shop-floor production and maintenance cycle

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 $\mu = 1$.

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, then it enters into MOM. Fig. 5 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. 8.

$$\Phi_{j,INIT}^{PMM} = \Omega_j^p \times \Delta\phi_{INC} + \phi_j^{m0} \tag{8}$$

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. 9.

$$\Phi_{j,k}^{PMM} = \Phi_{j,INIT}^{PMM} + k \times \Delta\phi_{INC} \tag{9}$$

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 $\mu = 1$. 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:

$$\Phi_{j,k+1}^{PMM} = \Phi_{j,k}^{PMM} - \nu_k \times \Delta\phi_{DEC} \tag{10}$$

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.

$$T_{avg}^{PMM} = \frac{1}{M}\sum_{j=1}^{M} T_j^{PMM} \tag{11}$$

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. 12.

$$T_{min}^{PMM} = \frac{M \times \Phi_{INIT}^{PMM}}{N \times \Delta\phi_{DEC}} \tag{12}$$

$$\zeta_{avg}^{PMM} = \frac{T_{avg}^{PMM} - T_{min}^{PMM}}{T_{min}^{PMM}} \tag{13}$$

Thus we can define ζ_{avg}^{PMM}, average production completion delay (APCD) by following Eq. 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. 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:

$$\Phi_{j,k+1}^{MOM} = \Phi_{j,k}^{MOM} - (\nu_k - \mu) \times \Delta\phi_{DEC} \tag{14}$$

By considering $\mu = 1$, Eq. 14 will reduces to Eq. 10. Here, $\Phi_{j,k+1}^{MOM}$ will correspond to the *pending maintenance work-load* 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 pending maintenance work-load* (APMW) per time-step per machine, χ_j^{MOM} (Eq. 15) and average PMW per machine per time-step, χ_{avg}^{MOM} (Eq. 16).

$$\chi_j^{MOM} = \frac{1}{K} \sum_{k=1}^{K} \Phi_{j,k}^{MOM} \tag{15}$$

$$\chi_{avg}^{MOM} = \frac{1}{M} \sum_{j=1}^{M} \chi_j^{MOM} \tag{16}$$

5.2. Robot-controller algorithms

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. 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. 6. It requires total number of shop-tasks and sum of their stimulus.

Algorithm 1: Self-regulated task-allocation based on AFM

1: **Input:** $AllTaskInfo, RobotPose, TaskRecords, DeltaDistance$
2: **Output:** $SelectedTaskID$
3: **comment**: Stage 1: Get sum of stimulus of all tasks
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: $DistToTask \leftarrow$ **CalculateTaskDistance(** $RobotPose, TaskPose$**)**
11: $TaskStimuli \leftarrow$ **CalculateTaskStimuli(** $DistToTask, TaskSensitization, TaskUrgency, DeltaDistance$**)**
12: $TaskStimuliSum \leftarrow TaskStimuliSum + TaskStimuli$
13: $TaskRecords \leftarrow$ **UpdateTaskRecords(** $TaskStimuli, DistToTask, TaskUrgency,$
14: $TaskSensitization$**)**
16: **end for**
17: $RandomWalkStimuli \leftarrow$ **CalculateRandomWalkStimuli(** $TotalTasks, TaskStimuliSum$**)**
18: $AllStimuliSum \leftarrow TaskStimuliSum + RandomWalkStimuli$
20: **comment**: Stage 2: Find probability of doing each task
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(
27: $TaskProbability$**)**
28: $TaskRecords \leftarrow$ **UpdateTaskRecords(**$TaskProbability$**)**
29: **end while**
30: **comment**: Stage 3: Draw a random-number to match TaskID
31: $RandomNum \leftarrow$
GetRandomNumber(0, **Max(**$TaskProbabilityRange$**))**
32: **while** $TaskID \leq TotalTasks$ **do**
33: $RangeStart \leftarrow$ **Min(**$TaskProbabilityRange(TaskID)$**)**
34: $RangeEnd \leftarrow$ **Max(**$TaskProbabilityRange(TaskID)$**)**
35: **if** $RandomNum \geq RangeStart$
&& $RandomNum \leq RangeEnd$ **then**
36: $SelectedTaskID \leftarrow TaskID$
37: **end if**
38: **end while**

Stage 2. In the second stage of our task-allocation algorithm, we find the probability of each task (including random-walk) based on Eq. 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 the above example, if the random-number becomes 37, our task-probability range checking function selects *Task2* since 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 2: Robot's learning and forgetting of tasks based on AFM

Input: $TaskRecords, LeranRate, ForgetRate, SelectedTaskID$
Output: Updated $TaskSensitisation \in TaskRecords$
for all $Task \in TaskRecords$ **do**
  $TaskID \leftarrow$ ID $\in Task$
  $TaskSensitization \leftarrow$ Sensitisation $\in Task$
  if $TaskID \equiv SelectedTaskID$ **then**
    $TaskSensitization \leftarrow$ **Max(**$1, (TaskSensitization + LeanRate)$**)**
  else
    $TaskSensitization \leftarrow$ **Min(**$0, (TaskSensitization - ForgetRate)$**)**
  end if
end for

Robot learning and forgetting algorithm:
Algorithm 4.2 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.

6. Local P2P communication model (LPCM)

From our understanding of different kinds of communication strategies of biological social systems, this is obvious that an effective LSLC strategy can greatly enhance the self-regulated MRTA. In fact, most researchers in the field of swarm robotics now acknowledge that LSLC is the one of the most critical components of a swarm robotic system, as global behaviours emerges from local interactions among the individuals and their environment.

In this study, we have used the concepts of pheromone active-space of ants to realize our simple LSLC scheme. As discussed in Sec. **??**, 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 distances. Thus both communication and sensing occurs within a small communication range[2]. 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, some researchers have argued that this range can be set at design time based on the capabilities of robots [22]. 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 [31]. In this study, we have followed the former approach as our robots do not have the precise hardware to dynamically vary their communication and sensing ranges.

6.1. General characteristics of LPCM

Our LPCM relies on the local P2P communications among robots. we have assumed that robots can communicate to its nearby peers within a certain communication radius, r_{comm} and they can sense tasks within another radius r_{task}. They 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 this r_{comm}. Similarly, when a robot comes within this r_{task} of a task, it can sense the status of this task. Although the communication and sensing range can be different based on robot capabilities, we have considered them same for the simplicity of our implementation.

Local communication can also give robots similar task information as in centralized communication. In this case, 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 two ways: by either using their sensory perception (e.g., camera) or doing local P2P communication with others.

We characterize our communication model in terms of three fundamental issues of communication [32].

1. Message content: *what to communicate?*
2. Communication frequency: *when to communicate?*
3. Target message recipients: *with whom to communicate?*

In a typical multi-robot system, message content can be categorized into two types: state of each individual robot and target task (goal) information [33]. 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 model by introducing the concept of dynamic subscription. In a traditional publish/-subscribe communication 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

[2] 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.

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 name-space [32] 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}.

6.2. Implementation algorithm

Algorithm 3: 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 *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 own *LocalTaskInfo* signal to peers
18: $EmissionBuffer \leftarrow TaskInfoDB$
19: **for all** peer $\in$ $RobotPeers$ **do**
20: **EmitLocalTaskInfoSignal(**peer, $EmissionBuffer$ **)**
21: **end for**
22: $RobotPeers \leftarrow 0$

Within the context of our self-regulated MRTA experiments under AFM, we have formalized the following three aspects of LPCM into an implementation algorithm.

1. Local sensing of peers and tasks.
2. Listening to the task-information peer signals.
3. Emitting local task-information signals to peers.

From Algorithm 5.1, 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 also empty. Upon sensing a task, robot determines the task's pose and urgency of that task (according AFM rules described in Sec. 4.3). This is not strictly necessary as this information can also be available from alternate sources, e.g. via communicating with a TPS which is discussed later.

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 textitLocalTaskInfo 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.

7. Experiments

We have designed a set of manufacturing shop-floor scenario experiments for validating the effectiveness of our AFM in producing self-regulated MRTA. The overall aim of this experiments is to compare the various properties of self-regulated MRTA under GSNC and LSLC strategies. These experiments are grouped into four series labelled using the following letters: A, B C and D. Series A and B experiments are done using a centralized communication system under GSNC strategy and Series C and D experiments are carried out using an emulated local P2P communication under LSLC stategty. Below we describe the observables, parameters and implementation of our experiments.

7.1. Observables

Plasticity: Self-regulated DOL can be characterised by plasticity and task-specialization, in both macroscopic and microscopic levels. Within manufacturing shop-floor 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 PMM. Similarity, when there would be low workload in case of 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 DOL is always accompanied with task-specializations of agents. That means that 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 be able to 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 manufacturing shop-floor scenario, we can count the percentage of robots specializes on each task in our experiments.

Quality of task-performance: As discussed in Sec. 5.1 we can measure the quality of MRTA from the APCD. It first calculates the ideal minimum production time and then finds 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 can measure APMW within the context of our manufacturing shop-floor scenario. This can show 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 urgency 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 shop-floor work-load data, we can confirm the presence of flexibility in MRTA.

Energy-efficiency: In order to characterize the energy-efficiency in MRTA we can 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. Here we can measure 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 on self-regulated MRTA. 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.

7.2. Parameters

Table 1 lists a set of essential parameters of our experiments. We intend to have a set-up that is relatively complex, i.e., with a high number of robots and tasks in a large area. The diameter

Table 1: Experimental parameters of Series A & B experiments

Parameter	Series A | Series B
Total number of robots (N)	8 | 16
Total number of tasks (M)	2 | 4
Experiment area (A)	2 m^2 | 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

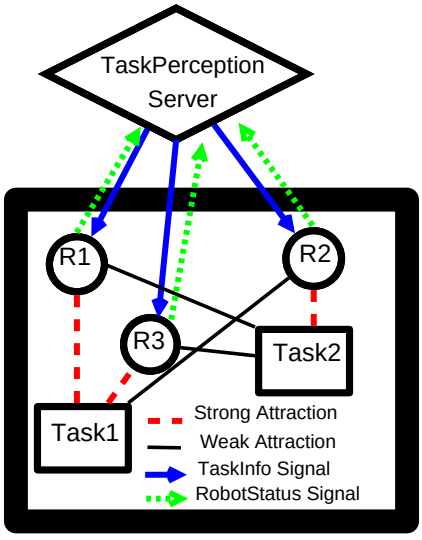


Figure 6: A centralized communication scheme

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. 8. We choose a limit of 0 and 1, where 0 means no urgency and 1 means maximum urgency. Same rule 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.

$$\Delta\phi_{INC} = \frac{\Delta\phi_{DEC} \times N}{2 \times M} \tag{17}$$

$$\Delta k_{DEC} = \frac{\Delta k_{INC}}{M - 1} \tag{18}$$

Eq. 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 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. 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.

Our local communication experiments follow the similar design of experimental parameters and observables of Series B experiments as outlined above. Table 2 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 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 "global sensing and global communication" of information by the robots, since in a large group of robots, it is not practical that a robot communicates with all other robots. On the other hand, Series C uses the half of the range 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.

Table 2: Experimental parameters of Series C & D experiments

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

7.3. Implementation

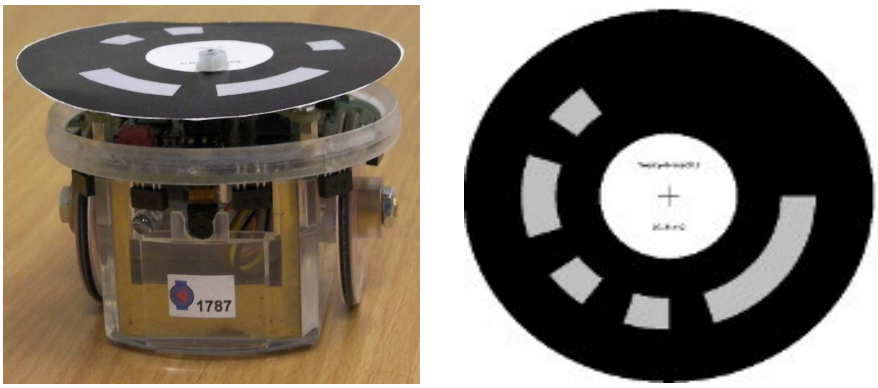
Ideally, AFM can be implementation as a complete distributed task-allocation system where each agent selects its own task based on its own external perception about task-urgencies (i.e. attractive fields), distances from tasks and internal task-sensitisation records. Such an implementation requires powerful robots with sophisticated sensors (camera, laser etc.) and sufficient computation and communication capabilities. In that case, 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 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 recalculate the task-urgencies and send them in next broadcast.

Fig. 6 shows how three robots are 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.

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 **??**, 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 pre-defined 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. **??**, 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 RCC for each physical robot. These RCCs also rely upon SwisTrack multi-robot



(a) E-puck robot (b) A binary-coded marker

Figure 7: (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.

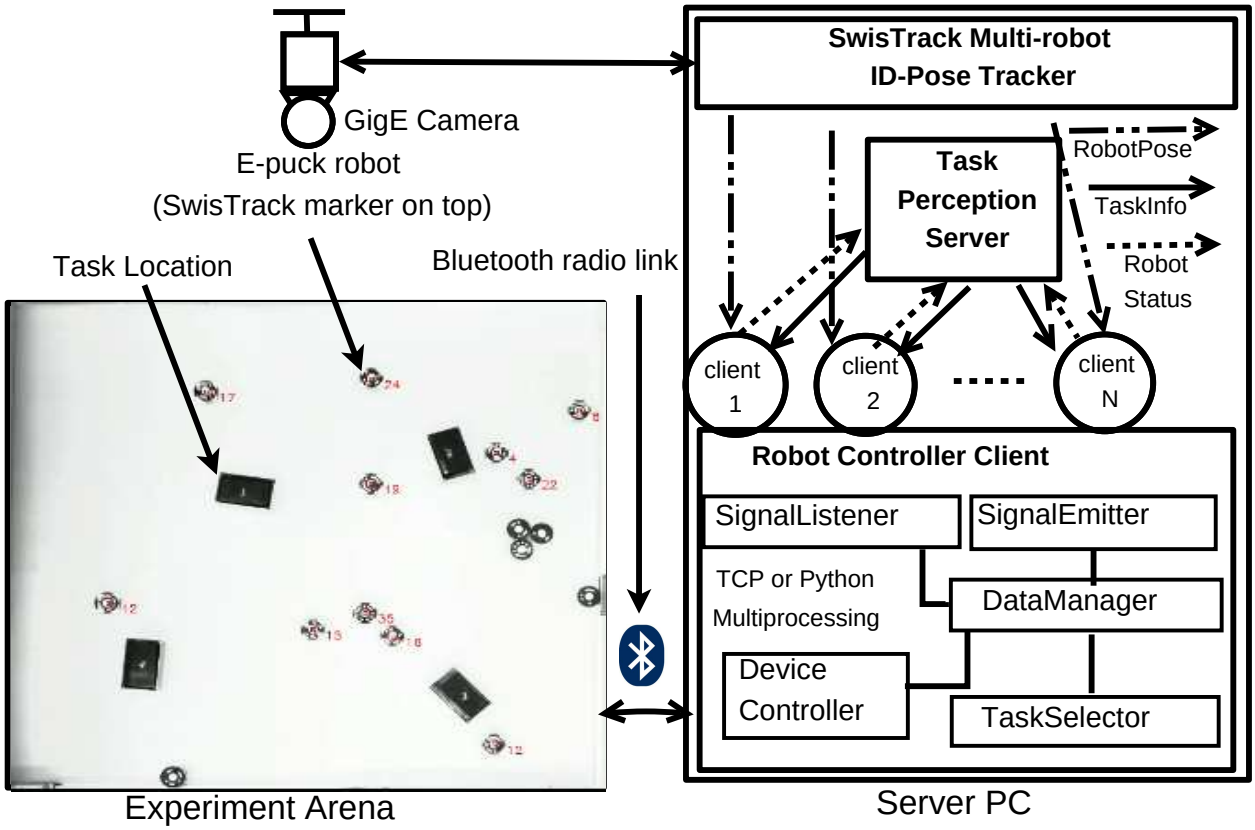
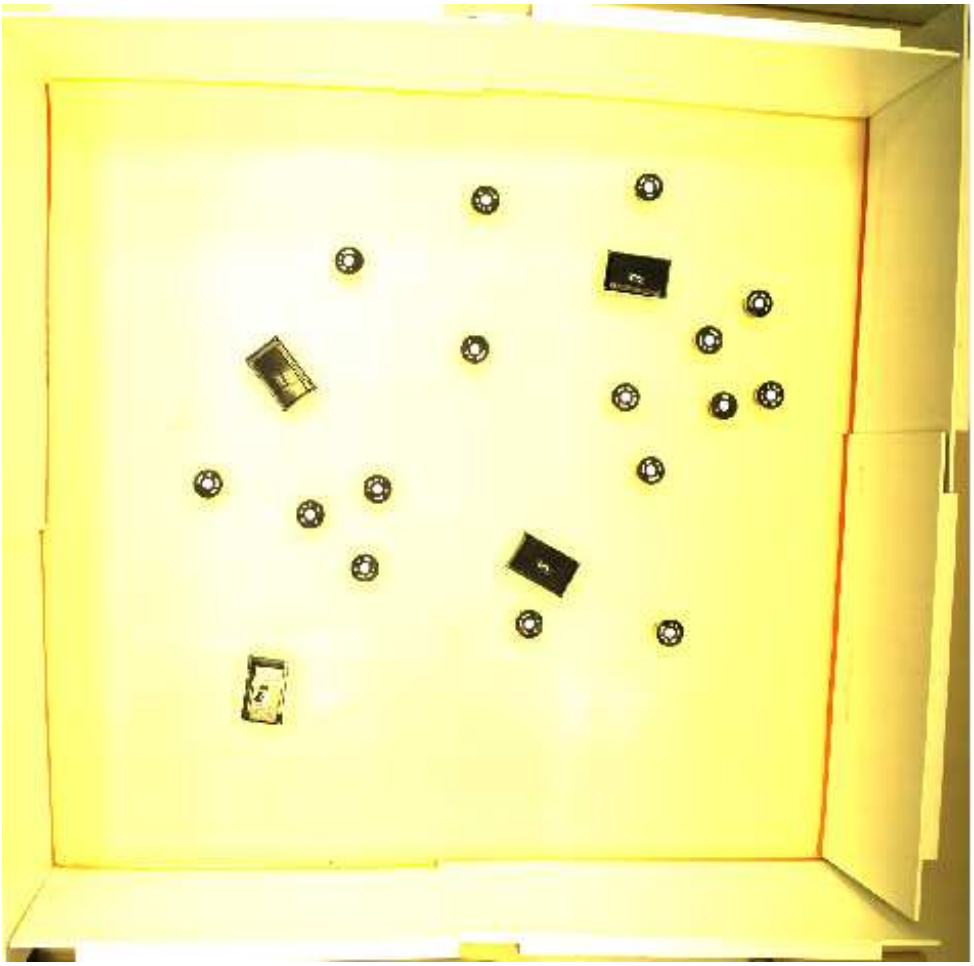


Figure 8: Hardware and software setup for series A & B experiments



(a)

(b)

Figure 9: Our experiment arena captured by (a) a GigE4900C camera mounted on 3m high ceiling (b) an ordinary camcorder.

Table 3: D-Bus signal interfaces among software components.

D-Bus Signal	Source	Destination	Payload
RobotPose	SwisTrack	RCC	Robot-pose x, y and oientation (theta).
TaskInfo	TPS	RCC	An array of all tasks' information each of them contains: task-id, time-stamp, task pose (x,y), task-urgency.
RobotStatus	RCC	TPS	Robot's id and its currently executing task's id.
RobotPeers	SwisTrack	RCC	List of IDs of peers of each robot within its communication range.
TaskNeighbour	SwisTrack	TPS	List of IDs of robots that are co-located within a task's perception range.
LocalTaskInfo	RCC	RCC	An array of known tasks' information.

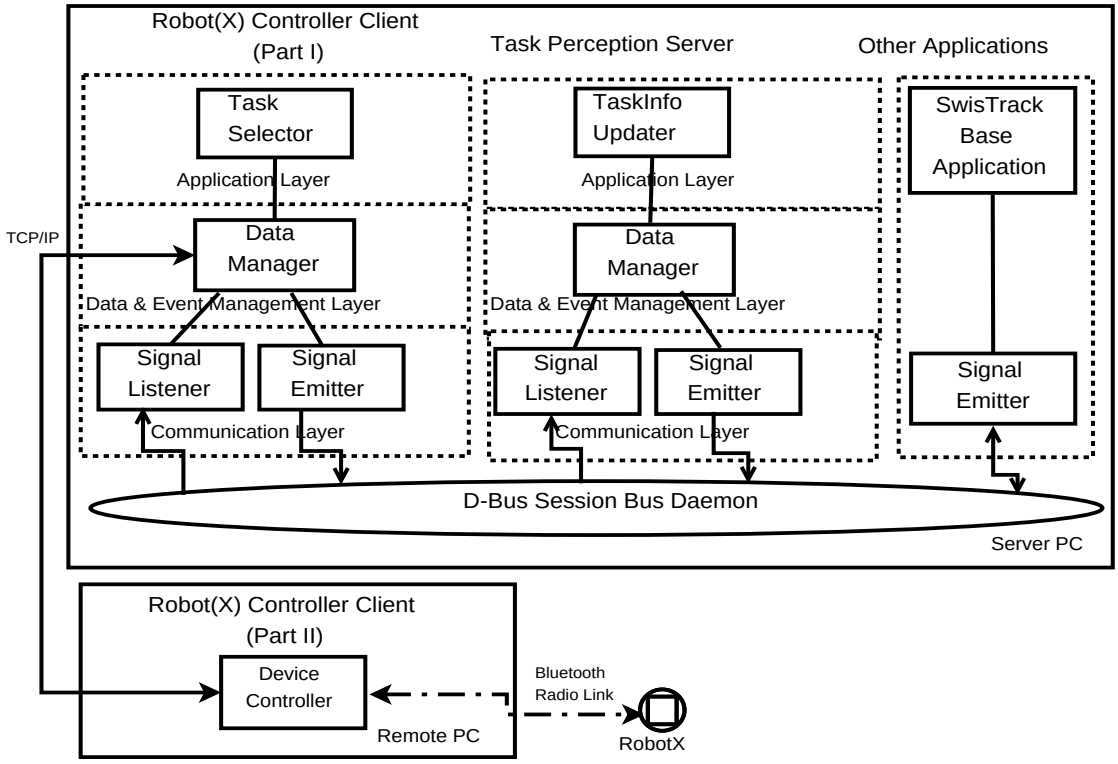


Figure 10: General outline of *HEAD*. A RCC application has been split into two parts: one runs locally in server PC and another runs remotely, e.g., in an embedded PC.

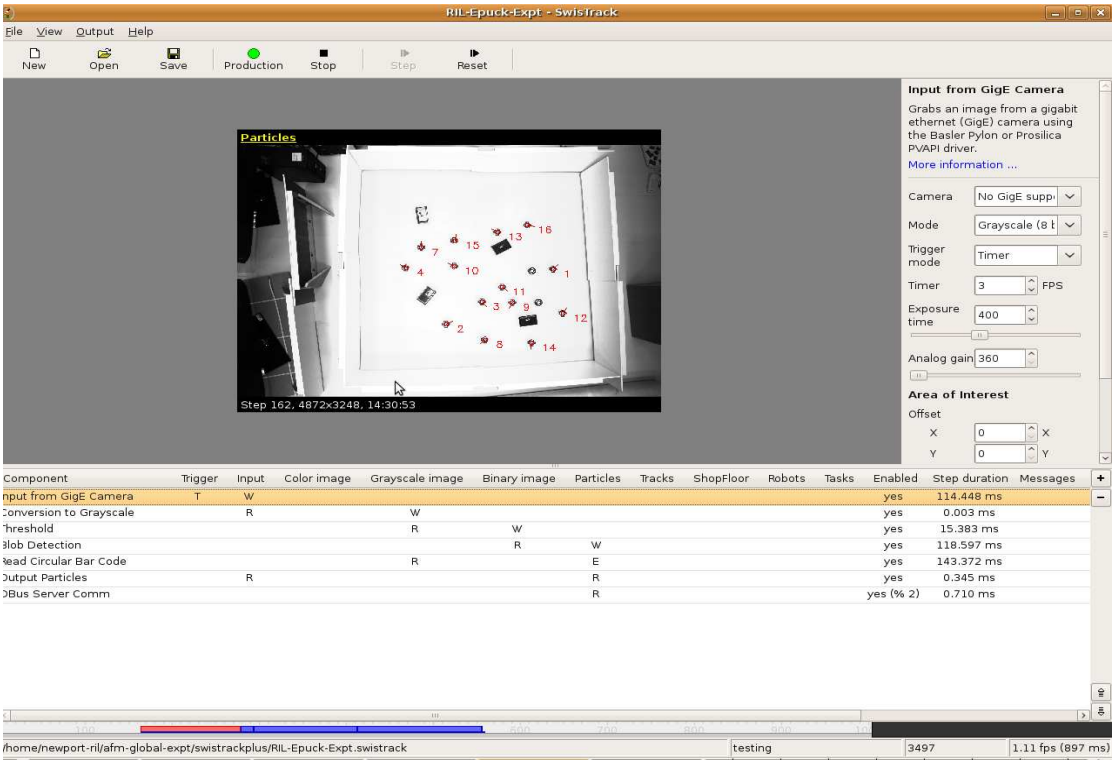


Figure 11: SwisTrack tracking a team of 16 robots under Ubuntu Linux 9.10 OS.

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 implementation. Below we describe the actual implementations of these components, i.e. D-Bus communication interfaces and detail implementations of TPS and RCC. Fig. 10 outlines the placements of various software components of our multi-robot control architecture based on their functional characteristics and processing requirements [34]. As shown in Fig. 8, each e-puck robot is controlled by a corresponding RCC. A RCC sends commands to the robot's firmware using BTCom protocol. 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[3].

In the implementation of RCC, 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 list the major data structures and corresponding event channels of `Data Manager`. 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. We have used Python *dictionary* type data

Table 4: Data structures and events used by the RCC

Data structure	Related event(s)
`mRobotID`	-
`mRobotPose`	`mRobotPoseAvailable`
`mTaskInfo`	`mTaskInfoAvailable`

`mSelectedTask`	`mSelectedTaskAvailable`
	`mSelectedTaskStarted`
	`mTaskTimedOut`

structure for all other data structures that are managed by the Python Multiprocessing's *Manager()* object. Any other process e.g. `TaskSelector`, can access `DataManager`'s data and event channels locally by taking a reference of `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.

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 func-

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

tions 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 signal'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.

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 executes a task, this event channel becomes activated by
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 mSelected TaskStarted event so that only one RobotStatus signal is emitted per task-cycle.

TaskSelector works in the application layer of our multi-robot control architecture. 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.

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 illustrated in Fig. 12. Externally we can observe two distinct physical state of a robot: it is either idle or in motion (separated by dotted 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* (i.e 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 (i.e device disconnected).

DeviceAvailable: DeviceController stays in this state until Data-Manager'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 (i.e. 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 *AtTask* state in next step. **AtTask:** In this state DeviceController discovers itself neat the task and normally until task time-out happens (i.e. task pending) it stays at this state.

We have added three new D-Bus signal interfaces in this version of our implementation. 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 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. 7.3).

LocalTaskInfo**:** This signal is emitted by RCCs to their peers

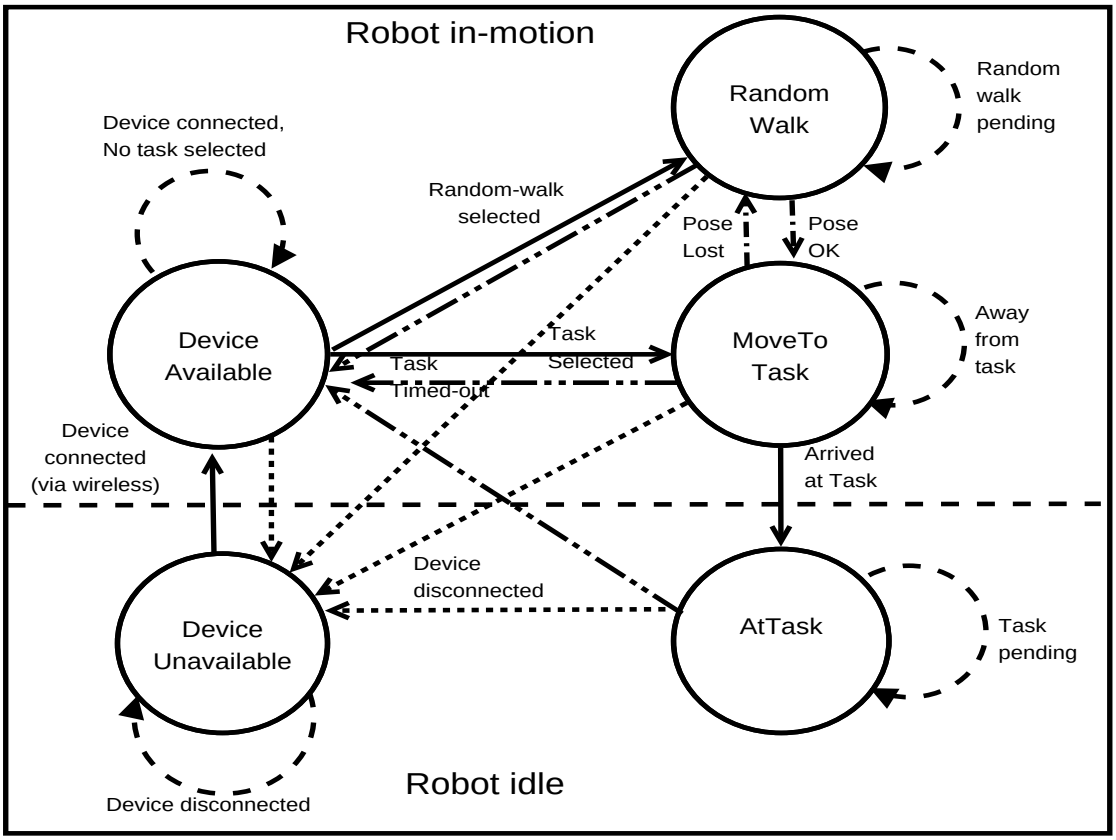


Figure 12: State diagram of `DeviceController` module

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.

As shown in Fig. **??**, 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 GitHub repository[4].

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 are done in the other parts of TPS implementation. The full Python implementation of TPS is available for download from the GitHub repository[5].

8. Results

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

[4] http://github.com/roboshepherd/EpuckDistributedClient
[5] http://github.com/roboshepherd/DitributedTaskServer

Shop-floor work-load history

In our experiments we have defined shop-floor work-load in terms of task urgencies. For example, Eq. 8 shows how we have calculated initial production work-load of our manufacturing shop-floor scenario. Fig. 8 show the dynamic changes in task-urgencies for the single iteration of each experiment. The fluctuations in these plots are resulted from the different levels of task-performance of our robots. In case of Series D, we can see that an unattended task, *Task4*, was not served by any robot for a long period and later it was picked up by some of the 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 M tasks at $(q+1)^{th}$ step:

$$\Delta\Phi_{j,q+1} = \sum_{j=1}^{M} (\phi_{j,q+1} - \phi_{j,q}) \tag{19}$$

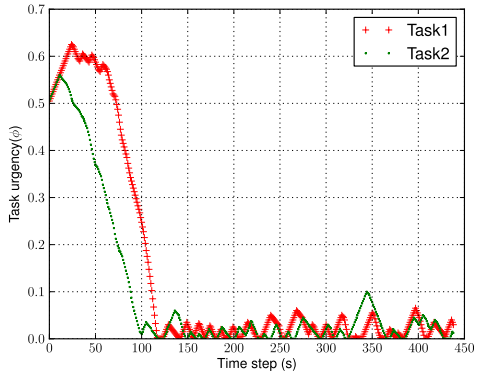
From Fig. 8 shows the dynamic shop-floor workload for all four series of experiments. From these plots, we can see that initially the sum of changes of task urgencies (shop-floor workload) is going towards negative direction. This implies that tasks are being served by a high number of robots.

Ratio of active workers

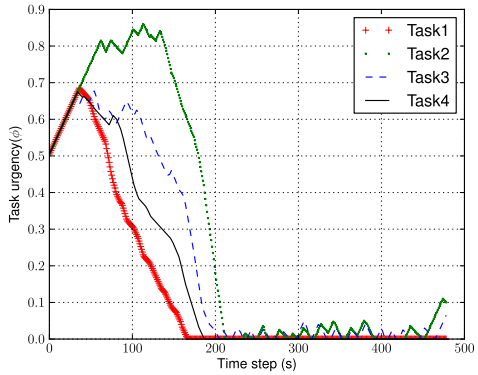
From Fig. 8, we can see that in production stage, when work-load is high, many robots are active in tasks. Here active workers ratio is the ratio of those robots that work on tasks to the total number of robots N of a particular experiment. Here we can see that this ratio varies according to the shop-floor work-load changes.

Shop-task performance

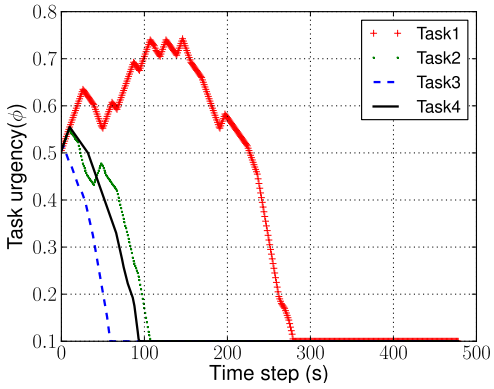
In our manufacturing shop-floor scenario, we have calculated the APCD and APMW as shown in Fig. 8. For Series A we have got average production completion time 111 time-steps (555s) where sample size is (5 x 2) = 10 tasks, SD = 10 time-steps (50s). According to Eq. 12, our theoretical minimum production 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. 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 (250s). 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. 13 gives us APCD, ζ = 2.3. For Series C we have got average production completion time 121 time-steps (605s) with SD = 36 time-steps (180s). For Series D, average production completion time is 123 time-steps (615s) with SD = 40 time-steps (200s). According to Eq. 12, our theoretical minimum production completion time is 50 time-steps (250s) as discussed in Sec **??**. The values of APCD are as follows. For



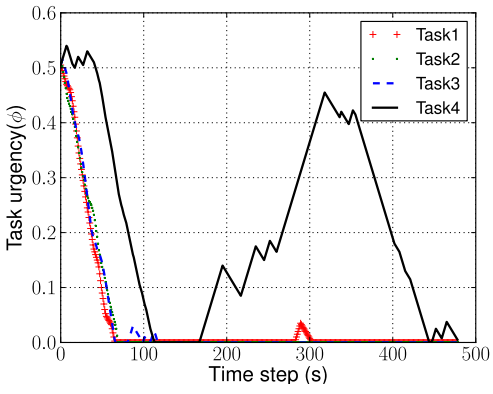
(a) Series A



(b) Series B

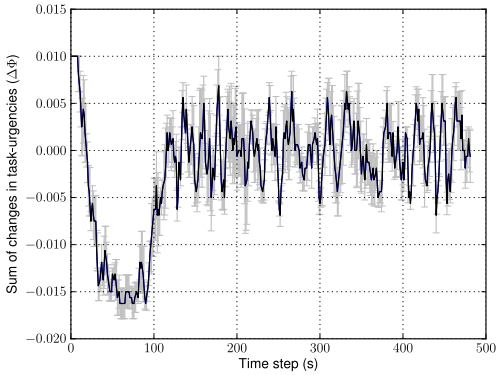


(c) Series C

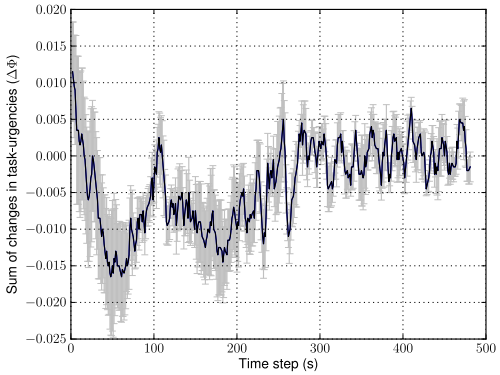


(d) Series D

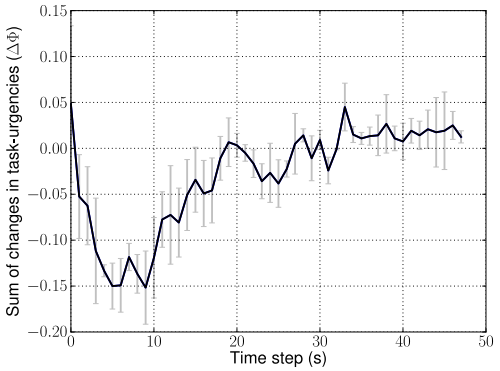
Figure 13: Changes in task-urgencies.



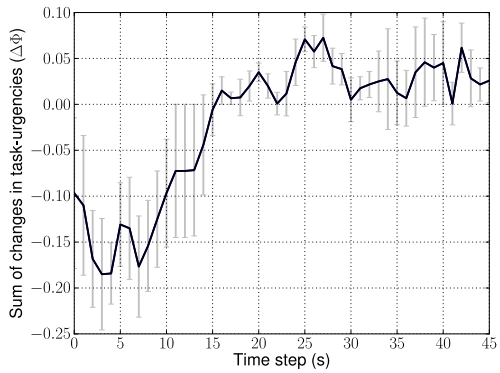
(a) Series A



(b) Series B

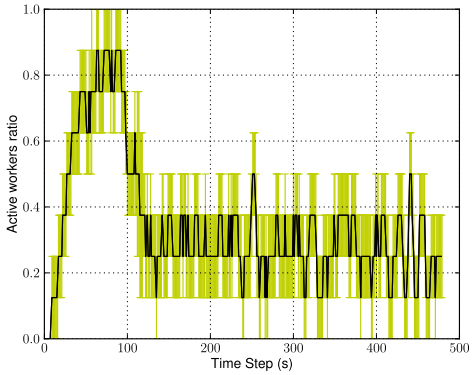


(c) Series C

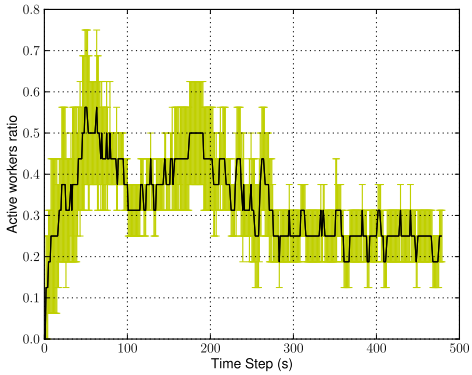


(d) Series D

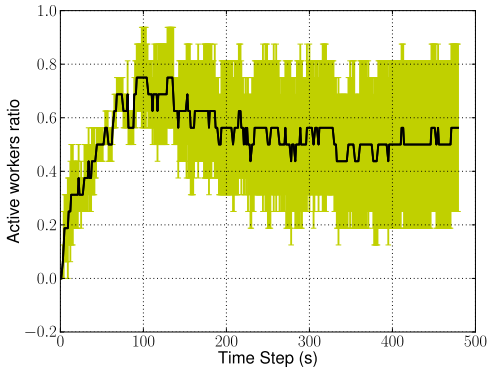
Figure 14: Shop-floor workload change history.



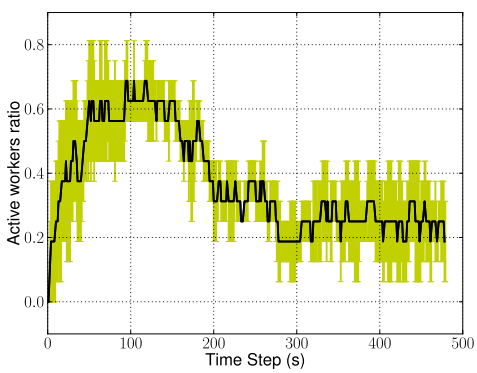
(a) Series A



(b) Series B

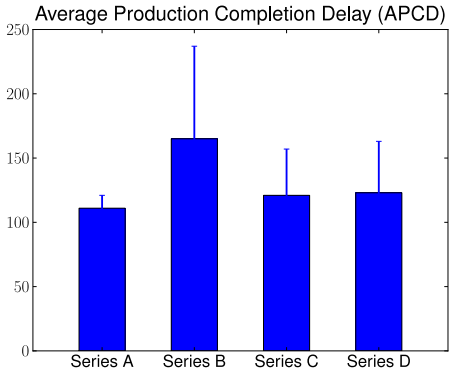


(c) Series C

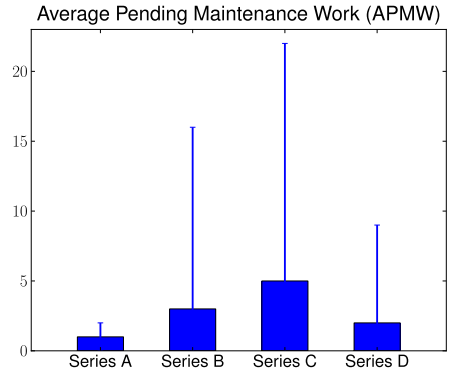


(d) Series D

Figure 15: Self-organized allocation of robots.



(a) APCD



(b) APMW

Figure 16: Performance of manufacturing shop-floor tasks.

Series C, $\zeta = 1.42$ and for Series D, $\zeta = 1.46$. For both series of experiments APCD values are very close.

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 ($\chi = 0.000235$) 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, $\chi = 0.012756$ 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. The APMW of Series C experiments give us an average time length of 359 time-steps (1795s). In this period we calculated APMW and it is 5 time-steps with SD = 17 time-steps and $\chi = 0.023420$. For Series D experiments, from the average 357 time-steps (1575s) of maintenance activity of our robots per experiment run, we have got APMW, $\chi = 0.005359$ which corresponds to the pending work of 2 time-steps (10s) where SD = 7 time-steps.

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 maximum sensitization value of a robot among all tasks during an experiment run. This value is then averaged for all robots using the following equation.

$$K^G_{avg} = \frac{1}{N}\sum_{i=1}^{N}\max_{j=1}^{M}\left(k^i_{j,q}\right) \tag{20}$$

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. 20 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.

$$Q^G_{avg} = \frac{1}{N}\sum_{i=1}^{N} q^i_{k=k_{max}} \tag{21}$$

In Eq. 21, $q^i_{k=k_{max}}$ 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 5 and Table 6 show the peak sensitization values of Series A and Series B experiments respectively. Based on Eq. 20 and Eq. 21, we have got the peak task-sensitization K^G_{avg} values: 0.40 (SD=0.08) and 0.30 (SD=0.03), and their respective time-step Q^G_{avg} values: 38 (SD=13) and 18 (SD=5) time-step. They are shown in Fig. **??** and Fig. **??**. Here we can see that the robots in Series A had higher chances of task-specialization

Table 5: Peak task-sensitization values of robots in a Series A experiment.

Robot ID	Maximum k	At time-step (q)	Task
1	0.54	64	Task1
4	0.32	14	,,
5	0.27	11	,,
3	0.47	63	Task2
2	0.46	64	,,
6	0.20	10	,,
7	0.18	4	,,
8	0.15	3	,,

Table 6: Peak task-sensitization values of robots in a Series B experiment.

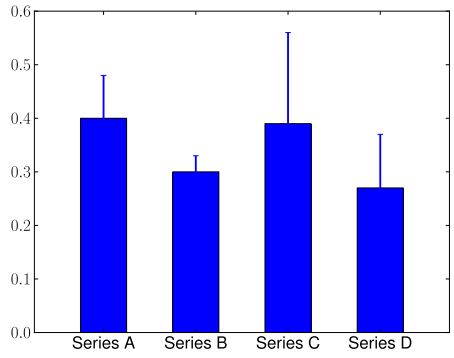
Robot ID	Maximum k	At time-step (q)	Task
24	0.41	29	Task1
13	0.31	19	,,
16	0.18	4	,,
1	0.64	66	Task2
35	0.34	12	,,
5	0.28	14	,,
22	0.18	20	,,
17	0.16	6	,,
3	0.14	12	,,
9	0.68	66	Task3
6	0.43	71	,,
15	0.19	4	,,
14	0.15	4	,,
31	0.14	4	,,
19	0.22	12	Task4
12	0.16	10	,,

Table 7: Peak task-sensitization values of robots in a Series C experiment.

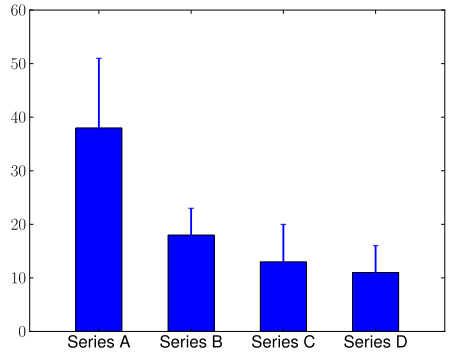
Robot ID	Maximum k	At time-step (q)	Task
16	1.00	52	Task1
12	0.58	24	,,
1	0.16	2	,,
13	0.13	1	,,
9	1.00	41	Task2
3	0.55	23	,,
19	0.24	6	,,
35	0.24	6	,,
15	0.13	1	,,
33	0.13	2	,,
6	0.61	29	Task3
31	0.59	35	,,
5	0.13	1	,,
17	0.36	10	Task4
20	0.09	1	,,
22	0.09	1	,,

Table 8: Peak task-sensitization values of robots in a Series D experiment.

Robot ID	Maximum k	At time-step (q)	Task
19	1.00	34	Task1
35	1.00	60	,,
24	0.41	16	,,
12	0.20	6	,,
1	1.00	41	Task2
31	1.00	47	,,
22	0.40	10	,,
9	0.38	12	,,
15	0.16	2	,,
33	0.13	1	,,
5	0.92	70	Task3
13	0.74	36	,,
17	0.76	62	Task4
6	0.22	4	,,
3	0.13	2	,,
16	0.13	1	,,



(a) Task-sensitization (K)



(b) Time-steps (Q)

Figure 17: Overall task-specialization of robot groups based on Peak task-sensitization values.

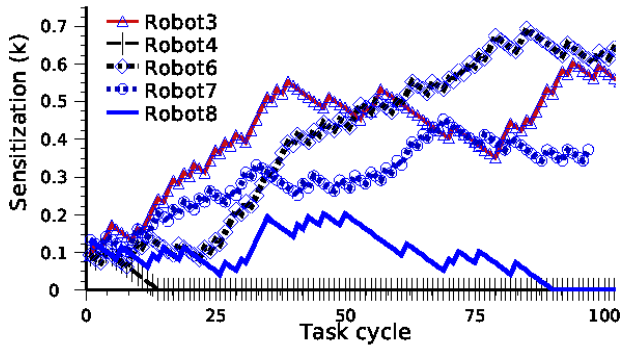


Figure 18: Task specialization on Task3 for a Series B experiment.

than that of Series B experiments. Fig. 18 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.

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.

$$\Delta U_{q+1} = \sum_{i=1}^{N} \delta u_{i,q+1} \tag{22}$$

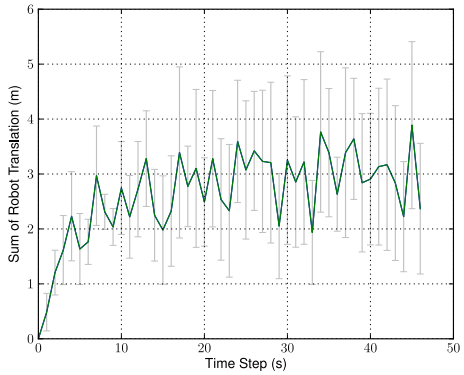
The sum of translations of robots from our experiments are plotted in Fig. 19. In this plot we can see that robot translations also vary over varying task requirements of tasks. In this plot we can see that robot translations also vary over varying task requirements of tasks.

Communication load

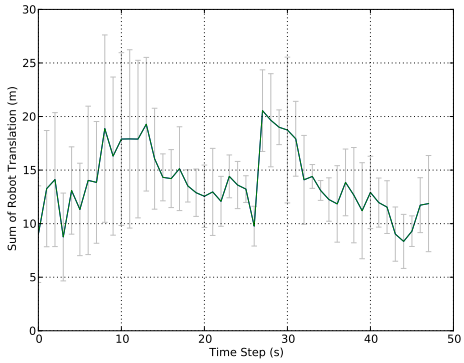
Fig. 8 (a) and (b) shows the number of received `TaskInfo` signals by each robot in Series A and Series B experiments. Since the duration of each time-step is 50s long and TPS emits signal in every 2.5s, there is an average of 20 signals in each time-step. The overall P2P task information signals of both of these local modes are plotted in Fig. 8 (c) and (d). As an example of P2P signal reception of a robot, Fig. 8 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.

9. Discussions

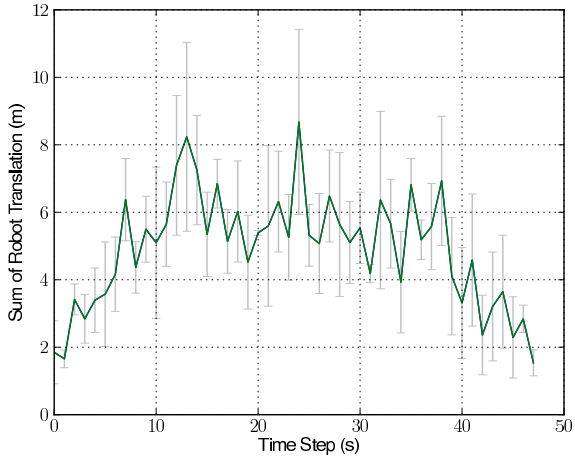
Self-regulated DOL. From our experimental results, we have noted several aspects of self-regulated DOL that exposes the power of AFM. As we have pointed out that this self-regulated DOL, as observed in biological and human social



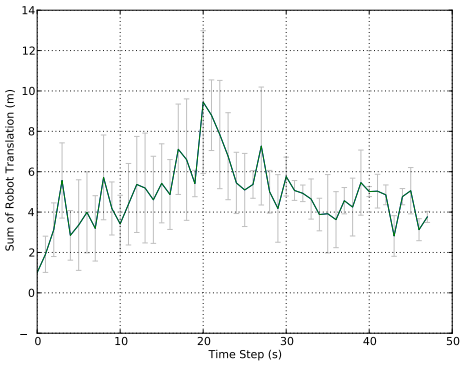
(a) Series A



(b) Series B

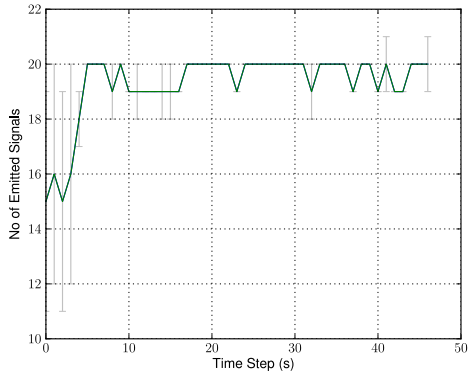


(c) Series C

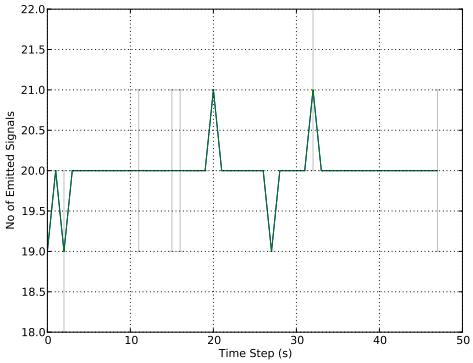


(d) Series D

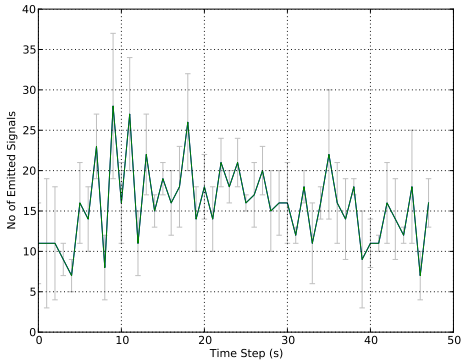
Figure 19: Sum of the translations of robots



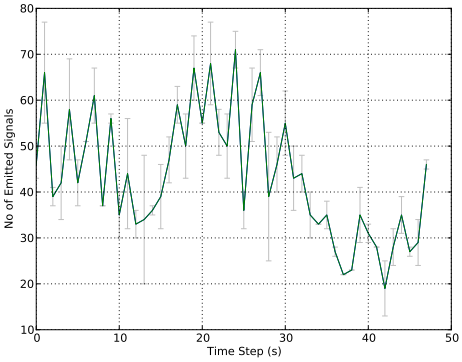
(a) Series A



(b) Series B

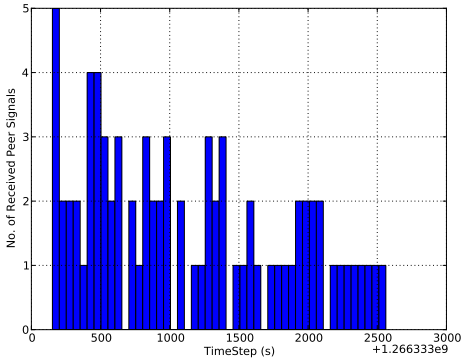


(c) Series C

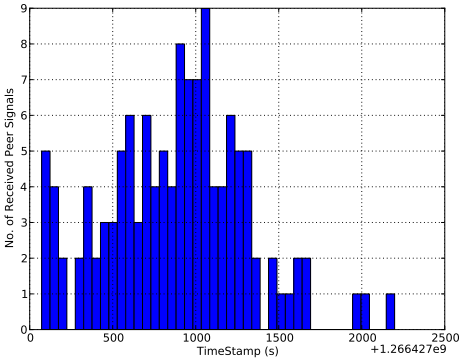


(d) Series D

Figure 20: Frequency of `TaskInfo` signalling of (a) and (b) by TPS and (c) and (d) by local peers.



(a) Series C



(b) Series D

Figure 21: `LoaclTaskInfo` (P2P) signals caught by Robot12 in (a) Series C and (b) Series D experiment

systems, needs to satisfy several important characteristics, e.g. plasticity, task-specialization. In addition to satisfying those basic qualities, AFM has demonstrated many other aspects. Our self-regulated robots, driven by AFM, effectively handle the dynamic work-load in our manufacturing shop-floor. They can dynamically support the need to work on currently demanding tasks, if there any. The variations of active worker ratio supports this.

From the self-organized worker allocations of AFM, it is clear to us that although in larger system (Series B) the degree of variations of active-worker ratio can show us significantly unpredictable patterns, nevertheless the self-regulated rules drive the robots to respond to the dynamic needs of the system. This means that AFM can sufficiently produce the plasticity of DOL in order to meets the dynamic work-load of the system.

Learning and Forgetting. From the individual and group-level task-
specialization, we can see that robots can maintain both task-specialization and flexibility. In a self-organized system, it is very common that only a few individuals specialize on tasks and others generally do not. From two samples data sets, we can see that in particular runs of Series A and Series B experiments, task-sensitization values of only 2-3 robots reach above the group-level average score. Thus in both types of experiments, robots exhibit similar task-specialization behaviours.

From task-sensitization we can also see that a limited number of robots are specialized in tasks. Thus most of the other robots are flexible in selecting any tasks as their task-specializations do not bind them to particular tasks.

Concurrency and robustness. As a consequence of fewer robots specializing in tasks, we can also see that robots can concurrently consider different tasks without being biased to a particular task all the time. Our experiments also show us the robust DOL as in case of both high and low work-loads present in the system. This is evident from the manufacturing shop-floor task performance during PMM and MOM. For example, in case of Series B experiments APMW was 13 time-steps (65s) which corresponded to pending work-load of 0.065 unit for a single robot. Thus, on an average, before the work-load exceeded by about 13 percent of initial work-load, robots were able to respond to a task.

Communication load. In these experiments we used a centralized communication system (source of attractive fields) that serves the robot with necessary task-perception information. Although our robot-controllers software RCC was also co-located in the same host-PC, they can be distributed to several PCs or robot's on-board PCs. Our centralized communication system has the advantage of minimising the communication load and the disadvantage of a single point of failure as well as single point of load. In the next chapter we present how the task perception can be decentralized by P2P communications among RCCs.

Scaling-up. We have observed the effect of scaling-up the robot team size. The system size of Series B is double of that of Series A in terms of robots, tasks and experiment arena. Keeping a fixed ratio of robot-to-task and task-to-arena we have intended to see the scaling effects in our experiments. Here we see that both systems can show sufficient self-regulated DOL, but task-performance of both systems varies significantly. For example, the value of APCD in Series B is higher by 1.08. This means that performance is decreased in Series B experiments despite having the resources in same proportion in both systems. This occurs partly due to the greater stochastic effects found in task-allocation in a larger system, e.g. presence of more tasks produce higher stochastic behaviours in robot's task selection.

Table 9: Sum of translations of robots in Series A-D experiments.

Series	Average translation (m)	SD
A	2.631	0.804
B	13.882	3.099
C	4.907	1.678
D	4.854	1.592

Similarly we can see that in larger system robots have less chances to specialize on tasks, as the Series B experiments show us that the overall average task-specialization of the group K^G_{avg} is lower by 0.10 and it lasts for significantly less time (the difference of Q^G_{avg} of both systems is 20 time-steps). Thus, in a large group, robots are more likely to switch among tasks more frequently and this produces more translation motions which cost more energy (e.g. battery power) in task-performance.

Comparisons between local communication and sensing strategies: Results from Series C and Series D experiments show us many similarities and differences with respect to the results of Series A and Series B experiments. Both Series C and Series D experiments show similar APCD values: 1.42 and 1.46 respectively, which are significantly less than Series B experiment result (APCD = 2.3) and are close to Series A experiment result (APCD = 1.22). This means that for large group, task-performance is efficient under LSLC strategy (Series C and Series D) comparing with their GSNC counterpart (Series B).

Besides, in terms of task-specialization, the overall task-specialization of group in Series C (K^G_{avg} = 0.4) is closer to that of Series A experiments (K^G_{avg} = 0.39) and interestingly, the value of Series D (K^G_{avg} = 0.27) is much closer to that of Series B experiments (K^G_{avg} = 0.30). So task-specialization in large group under LSLC strategy shows higher performance than their GSNC counter part. Besides task-specialization happens much faster under LSLC strategy as we can see that the average time to reach peak sensitization values of the group, Q^G_{avg} in Series C is lower than that of Series A values by 25 time-steps.

From the robot motion profiles found in all four series of experiments, we have found that under LSLC strategy, robot translations have been reduced significantly. Table 9 summarizes the average translations done by robots in all four series of experiments. From this table we can see than Series C and Series D show about 2.8 times less translation than that in Series B experiments. The translation of 16 robots in Series C and Se-

ries D experiments are approximately double (1.89 times) than that of Series A experiments with 8 robots. Thus the energy-efficiency under LSLC strategy seems to be higher than that under GSNC strategy.

From the above results we can see that large group robots achieve better MRTA under LSLC strategy. The local sensing of tasks prevents them to attend a far-reaching task which may be more common under global sensing strategy. However, as we have seen in Fig. **??** some tasks can be left unattended for a long period of time due to the failure to discover it by any robot. For that reason we see that the values of APMW is slightly higher under LSLC strategy. But this trade-off is worth as LSLC strategy provides superior self-regulated MRTA in terms of task-performance, task-specialization and energy-efficiency.

10. Conclusions

The benefits of deploying a large number of robots in dynamic multi-tasking environment can not be fully realized without adopting an effective communication and sensing strategy for a given multi-robot system. This study has focused on comparing two bio-inspired communication and sensing strategies in producing self-regulated MRTA by an interdisciplinary model of DOL, AFM. Under the GSNC strategy, AFM has produced the desired self-regulated MRTA among a group of 8 and 16 robots. This gives us the evidence that AFM can successfully solve the MRTA issue of a complex multi-tasking environment like a manufacturing shop-floor. Under the LSLC strategy, AFM can also produce the desired self-regulated MRTA for 16 robots with different communication and sensing ranges.

From our comparative results, we can conclude that for large group of robots, degradation in task-performance and task-specialization of robots are likely to occur under GSNC strategy that relies upon a centralized communication system. Thus GSNC strategy can give us better performance when the number of tasks and robots are relatively small. This confirms us the assertions made by some biologists that self-regulated DOL among small group of individuals can happen without any significant amount of local communications and interactions. However, our findings suggest that task-specialization can still be beneficial among the individuals of a small group which contradicts the claim that small groups only posses the generalist workers, but not the specialists.

On the other hand, LSLC strategy is more suitable for large group of individuals that are likely to be unable to perform global sensing and global communications with all individuals of the group. The design of communication and sensing range is still remained as a critical research issue. However, our results suggest that the idea of maximizing information gain is not appropriate under a stochastic task-allocation process, as more information causes more task-switching behaviours that lowers the level of task-specialization of the group. This might not be the case under a deterministic task-allocation scheme where more information leads to better and optimum allocations, but that is limited to a small group of individuals. Nevertheless, despite having the limited communication and sensing range, LSLC strategy helps to produce comparatively better task-allocation with increased task-specialization and significantly reduced motions or savings in energy consumption.

References

[1] R. C. Arkin, Behavior-based robotics, Cambridge, Mass. : MIT Press, c1998., 1998, includes bibliographical references (p. [445]-476) and indexes.

[2] L. E. Parker, F. Tang, Building multirobot coalitions through automated task solution synthesis, Proceedings of the IEEE 94 (2006) 1289–1305.

[3] B. P. Gerkey, M. J. Mataric, A formal analysis and taxonomy of task allocation in multi-robot systems, The International Journal of Robotics Research 23 (2004) 939.

[4] A. B. Sendova-Franks, N. R. Franks, Self-assembly, self-organization and division of labour', Philosophical Transactions of the Royal Society of London B 354 (1999) 1395–1405.

[5] B. Gerkey, M. Mataric, Multi-robot task allocation: analyzing the complexity and optimality of key architectures, Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on 3.

[6] L. E. Parker, Distributed intelligence: Overview of the field and its application in multi-robot systems, Journal of Physical Agents, special issue on multi-robot systems vol. 2 (no. 2) (2008) 5–14.

[7] L. Chaimowicz, M. F. M. Campos, V. Kumar, Dynamic role assignment for cooperative robots, Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on 1.

[8] M. B. Dias, R. M. Zlot, N. Kalra, A. Stentz, Market-based multirobot coordination: A survey and analysis, Proceedings of the IEEE 94 (2006) 1257–1270.

[9] K. Lerman, C. Jones, A. Galstyan, M. J. Mataric, Analysis of dynamic task allocation in multi-robot systems, The International Journal of Robotics Research 25 (2006) 225.

[10] C. R. Kube, H. Zhang, Collective robotics: From social insects to robots, Adaptive Behavior 2 (1993) 189.

[11] W. Liu, A. F. T. Winfield, J. Sa, J. Chen, L. Dou, Towards energy optimization: Emergent task allocation in a swarm of foraging robots, Adaptive Behavior 15 (3) (2007) 289–305.

[12] E. Arcaute, K. Christensen, A. Sendova-Franks, T. Dahl, A. Espinosa, H. J. Jensen, Division of labour in ant colonies in terms of attractive fields.

[13] R. Jeanne, Group size, productivity, and information flow in social wasps, Information processing in social insects (1999) 3–30.

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

[15] W. Shen, D. H. Norrie, J.-P. Barthes, Multi-agent systems for concurrent intelligent design and manufacturing, Taylor & Francis, London, 2001.

[16] P. Stone, M. Veloso, Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork, Artificial Intelligence 110 (1999) 241–273.

[17] C. Belta, V. Kumar, Abstraction and control for groups of robots, IEEE Transactions on Robotics 20 (5) (2004) 865–875.

[18] G. Pereira, V. Kumar, M. Campos, Decentralized algorithms for multi-robot manipulation via caging, Algorithmic Foundations of Robotics V (2003) 257–274.

[19] T. Dahl, M. Mataric, G. Sukhatme, Distributed multi-robot task allocation through vacancy chains, Autonomous Robots.

[20] E. Bonabeau, M. Dorigo, G. Theraulaz, Swarm intelligence: from natural to artificial systems, Oxford University Press, 1999.

[21] M. Krieger, J. Billeter, The call of duty: Self-organised task allocation in a population of up to twelve mobile robots, Robotics and Autonomous Systems 30 (1-2) (2000) 65–84.

[22] W. Agassounon, A. Martinoli, Efficiency and robustness of threshold-based distributed allocation algorithms in multi-agent systems, in: Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 3, ACM, 2002, p. 1097.

[23] C. Jones, M. Mataric, Adaptive division of labor in large-scale minimalist multi-robot systems, in: 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings, Vol. 2, 2003.

[24] N. Kalra, A. Martinoli, A comparative study of market-based and threshold-based task allocation, Distributed Autonomous Robotic Systems 7 (2007) 91–101.
[25] B. B. Werger, M. J. Mataric, Broadcast of local eligibility for multi-target observation, Distributed Autonomous Robotic Systems 4 (2001) 347356.
[26] S. Botelho, R. Alami, T. LAAS-CNRS, M+: a scheme for multi-robot cooperation through negotiated taskallocation and achievement, Proceedings IEEE International Conference on Robotics and Automation 2.
[27] B. P. Gerkey, M. J. Mataric, Sold!: Auction methods for multirobot coordination, IEEE Transaction on Robotics and Automation 18.
[28] R. Zlot, A. Stentz, M. Dias, S. Thayer, Multi-robot exploration controlled by a market economy, Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on 3.
[29] H. Çelikkanat, A. Turgut, E. Şahin, Guiding a robot flock via informed robots, Distributed Autonomous Robotic Systems 8 (2008) 215–225.
[30] S. Camazine, N. Franks, J. Sneyd, E. Bonabeau, J. Deneubourg, G. Theraula, Self-organization in biological systems, Princeton, N.J. : Princeton University Press, c2001., 2001, what is self-organization? – How self-organization works – Characteristics of self-organizing systems – Alternatives to self-organization –.
[31] E. Yoshida, T. Arai, Performance analysis of local communication by cooperating mobile robots, IEICE Transactions on Communications 83 (5) (2000) 1048–1059.
[32] B. Gerkey, M. Mataric, Principled communication for dynamic multi-robot task allocation, Experimental Robotics VII (2001) 353–362.
[33] T. Balch, Communication, diversity and learning: Cornerstones of swarm behavior, in: E. Sahin, W. Spears (Eds.), Swarm Robotics Workshop: State-of-the-art Survey, no. 3342 in Lecture Notes in Computer Science, Springer-Verlag, Berlin Heidelberg, 2005, pp. 21–30.
[34] M. Sarker, T. Dahl, Flexible Communication in Multi-robotic Control System Using HEAD: Hybrid Event-driven Architecture on D-Bus, in: In Proc. of the UKACC International Conference on Control 2010 (CONTROL 2010), to appear, 2010.