Robotic Clusters: Multi-Robot Systems as Computer Clusters

A topological map merging demonstration

Ali Marjovi [*1], Sarvenaz Choobdar [#2], Lino Marques [*3]
[*] Institute of Systems and Robotics, University of Coimbra, Portugal
[#] Department of Computer Science, University of Porto, Portugal
[1]ali@isr.uc.pt, [2]sarvenaz@dcc.fc.up.pt, [3]lino@isr.uc.pt

***Abstract*—In most multi-robot systems, an individual robot is not capable to solve computationally hard problems due to lack of high processing power. This paper introduces the novel concept of *robotic clusters* to empower these systems in their problem solvings. A robotic cluster is a group of individual robots which are able to share their processing resources, therefore, the robots can solve difficult problems by using the processing units of other robots. The concept, requirements, characteristics and architecture of robotic clusters are explained and then the problem of "topological map merging" is considered as a case study to describe the details of the presented idea and to evaluate its functionality. Additionally, a new parallel algorithm for solving this problem is developed. The experimental results proved that the robotic clusters remarkably speedup computations in multi-robot systems. The proposed mechanism can be used in many other robotic applications and has potential to increase the performance of multi-robot systems especially for solving problems that need high processing resources.**

***Index Terms*—Robotic Cluster, Computer Cluster, Distributed Robotics, Task Sharing in Multi-Robot Systems, Cluster Computing, High Performance Computing.**

I. INTRODUCTION

Multi-robot systems can potentially provide several advantages over single robot systems, namely higher task accomplishment speed, higher accuracy and fault tolerance. For example, multiple robots can localize themselves more efficiently [1], fulfill search and exploration missions faster [2], [3], [4], [5], and generate maps of unknown environments more accurately [6]. As argued in [7], performance/cost ratio is the main advantage of multi-robot systems over single-robot systems. Using heterogeneous robots with a subset of the capabilities to accomplish a particular task, one can use simpler robots that are less expensive to engineer than a single monolithic robot with all of the necessary capabilities. Therefore, multi-robot systems mostly consist of simple robots that usually have low computational capability due to cost constraints [8], [9]. However, there are situations where high computational capabilities are required to solve complex problems. This paper proposes a novel approach based on computer cluster concepts to increase the processing efficiency of each individual robot inside a distributed robotic system. The general idea is to design a corporation mechanism for high performance computing in multi-robot systems and solving complex problems in robotics applications.

There are two different architectures for multi-robot systems; centralized and decentralized (also known as distributed) architectures. In both, higher efficiency in performing the tasks is achieved by dividing the main task to a set of subtasks and distributing the subtasks between the individual robots. In the centralized approaches, a base station is responsible for distributing the subtasks between the robots whereas decentralized architectures lack such base station. It is widely claimed (e.g. in [10], [11], [12]) that decentralized architectures have several inherent advantages over centralized architectures, including fault tolerance, reliability, and scalability [13]. Moreover, there are hybrid centralized/decentralized architectures ([14], [15]) wherein there is a central planner that applies high-level control over robots which have some degree of autonomy in their task distribution.

In multi-agent systems (MAS), the sub-problems of a constraint satisfaction problem are allowed to be subcontracted to different problem solving agents with their own interests and goals. Distributing a number of independent subtasks on a given number of non identical agents is an area that has attracted much interest in the past years, especially from operations research [16], [17], [18], artificial intelligence [19], [20] and also game theory [21]. Distribution and scheduling of subtasks seeks algorithms that minimize the total time it takes for all subtasks to be completed. This problem has been well structured for the job shop related applications [21]. Moreover, interaction protocols between the agents [22], planning communicative actions [23], [24], agents' competitiveness [25], types of negotiations [26], agents' commitments [27] and resource managements [28] are among the important challenges in MAS that are mostly common challenges in several robotic applications as well. Although much of the research in non-robotic MAS is relevant to robotics, this paper studies the processing ability of the individuals in multi-robot systems that is very different from MAS. Despite the robots in multi-robot systems that have their individual physical processing units, the agents in a MAS are usually virtual identities with a common processing unit. Therefore this paper focuses exclusively on distributed multi-robot systems.

Parallelism in multi-robot systems in particular is achieved by similar methodologies defined in MAS [19]. Robots usually run a distributed algorithm and based on a given criteria, the tasks are assigned among the robots. In these systems, usually the individual robots do not have a global view of the problem and they pick up the subtasks based on different strategies (e.g., market-based approaches [29], priority approaches [30] or coalitions [31], [32]). Each one of them solves its local problem and the emergent result will be the global solution.

Basically, in multi-robot systems, robots share the physical subtasks in order to increase the efficiency of the team [33], [34]. None of the studies in this field has ever suggested to share the processing resources of agents/robots to increase the processing efficiency of each individual in the system. However, there are several robotic applications which require high processing resources for the individual robots in a multi-robot system, namely “mapping” [35], [36], “robotics vision” [37], “path planning” [38] and “large-scale signal processing” [39]. For example, in multi-robot exploration and mapping, usually robots share their local topological maps and each one of them merges them to generate a global map of the whole environment. This problem is computationally hard if the robots do not share a reference coordinate frame in complex environments. In fact, topological map merging in this case is an NP-hard “maximum common subgraph isomorphism” problem. One of the best solutions for this problem in robotics with several simplifications and assumptions is an $O(n^4 log(n))$ algorithm [6], i.e. if the number of vertexes is several hundred, most robots are not able to merge two maps in real time. Considering the fact that robots usually have to merge more than two maps (based on the number of robots participating in the mission) and the low processing capability of the usual robots, this problem poses a tough challenge. There are many similar applications in this field that require high processing resources for individual robots.

Most of distributed robotic systems tend to use simple robots with advanced communication capabilities (e.g. [40]). Nowadays, small robotic processing boards run Linux and come with embedded wireless LAN adapters (e.g. Gumstix[1]). Utilizing robots' communication capabilities, we propose a corporation mechanism based on computer cluster concepts to facilitate problem solving in multi-robot systems.

A computer cluster is a type of distributed computer system which consists of a collection of networked computers working together as a single integrated computing resource [41]. The components of a cluster are commonly connected to each other through a local area network. Each individual computer called *node* contains one or more processors, RAM, hard disk and LAN card. Clusters are usually deployed to improve performance and availability, while typically being much more cost-efficient than single computers of comparable speed or availability. The advantage of clusters over other traditional platforms were proved by several academic projects such as Beowulf[2], Berkeley NOW [42], NetSolve [43] and HPVM[3]. These advantages include low-entry costs to access supercomputing-level performance, an incrementally upgradeable system, an open source development platform, and not being locked into particular vendor products. Nowadays, clusters have conquered not only the traditional science and engineering marketplaces for research and development, but also marketplaces of commerce and industry due to the overwhelming price/performance advantage of this type of platform over other ones.

The main contribution of this paper is to propose computer cluster concepts for multi-robot systems. The robots are able to establish a computer cluster and share their processing resources (brains) in solving complex problems when needed. To the best of the authors' knowledge, none of the previously designed multi-robot systems in robotics community has ever used the computer cluster concepts in order to increase the robots' processing capability. Additionally, this paper provides a novel parallel solution for the problem of topological map merging based on the proposed method. A state of the art of topological map merging is provided in section IV.

The term “robotic cluster” has been used in the context of robotics with different meanings. Several researchers have used cluster computers to process the data provided by a robot (mostly in image processing) such as [44]. However, this is very different from the presented concept in this paper. In the current study the robots do not *use* a computer cluster but they share their own processing resources to *establish* a computer cluster and solve a problem. Other researchers have used this term in modular robotics field, where robots join together and physically establish a cluster. In these studies “cluster” is defined as group of robots that physically attach to each other and generate a bigger structure (e.g. [45] and [46]). In a few works this term is used for the robots which actually do “clustering” tasks and literally means robots which classify and distinguish physical items [47]. Finally, “robotic cluster” in some cases is used to refer to a community of researchers who work on robotics. This paper represents a new definition for “robotic clusters”.

Section II presents a definition for robotic clusters and describes their characteristics and requirements and clarifies their differences with computer clusters. Section III presents a real implementation of robotic clusters. Afterwards, an efficient parallel solution to solve the problem of topological map merging (as a case study) is presented in section IV and the results obtained by a real cluster are presented. Finally, a discussion and conclusion is written in sections V.

II. Robotic clusters

In this system, each robot is a computation node of a computer cluster in order to empower multi-robot systems in solving computationally hard problems. Computer clusters are typically independent computers connected via a single local area network (LAN). Robots can also establish a computer cluster through wireless networks to form a ***robotic cluster***. We propose the following definition for a robotic cluster and then explain its characteristics, requirements and implementations in details.

[1]http://www.gumstix.net
[2]http://www.beowulf.org/
[3]http://www-csag.ucsd.edu/projects/hpvm.html

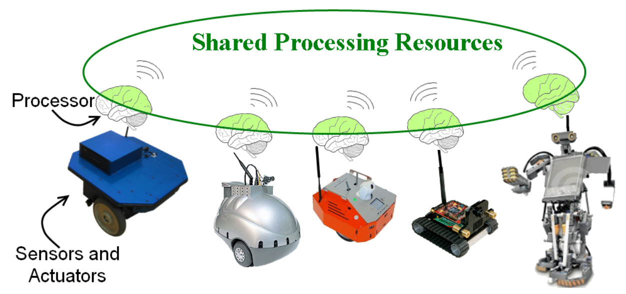


Fig. 1. Concept of robotic clusters

Definition 1. ***A* robotic cluster *is a group of individual robots which are able to share their processing resources among the group in order to quickly solve computationally hard problems.***

In a robotic cluster, the processing resources can be shared by applying computer clustering approaches. In these systems, each node is still able to run its own tasks independently, moreover, the CPUs of these robots are shared in the cluster. Therefore, when a robot sends some processing jobs to the others, the robots simultaneously run their own tasks and also the shared requested job. Fig. 4 demonstrates the concept of a robotic cluster which consists of multiple heterogeneous robots connected through a wireless network.

Applications of robotic clusters are tasks that demand

- extreme processing resources, and
- relatively cheap designs.

Processing high amount of data or mapping a large and complex environment with multiple simple robots are two examples of these applications. Processing and cost, both, are hard constraints that emphasize simplicity of the individual robots, and thus motivate a cluster approach to solve computationally hard problems. Excessive research is needed to find methodologies that allow designing and implementing robotic clusters to address several challenges in robotics research. This section goes to the details of this concept.

A. *Characteristics of robotic clusters*

Robotic clusters benefit from a number of advantages offered by concepts of computer clusters, including:

I. Processing Power: Each robot in a cluster is potentially able to use the processors of other robots. Most of the previous experiments with real robots show that the robots usually perform simple tasks (e.g. data logging and navigation) that do not require high processing power, but in some situations they need to process a huge amount of data (e.g. when a robot needs to process sensory data or finds itself in a different situation or needs to make a hard decision). Therefore in a group of robots, most of the robots usually perform simple tasks and only few robots need high processing resources. Using the clustering concepts the robots which need processing power will be able to use the processing units of the other robots. This idea can be more developed such that there can exist some special robots that do not participate directly in robotic mission but help the other robots with sharing their processing power.

II. Reduced Cost: The price of off-the-shelf simple computers or small laptops has decreased in recent years. These simple computers can be used as the processing unit of each individual robot and as a cluster they provide high processing capacity. Generally, in comparison with single robots systems, the parallel processing power of a robotic cluster is, in many cases, more cost efficient than a single robot with the same processing capability.

III. Scalability: One of the greatest advantages of robotic clusters is the scalability that they offer. While complex single robots with high processing resources have a fixed processing capacity, robotic clusters are easily expandable by adding additional nodes to the system when conditions change.

IV. Availability: When a single supper robot fails, the entire mission fails. However, if a robot in a robotic cluster fails, its operations can be simply transferred to another robot within the cluster, ensuring that there is no interruption in the service.

B. *Required equipments for robotic clusters*

For implementing a robotic cluster, there is no need of extra hardware rather than a wireless network and some robots equipped with simple computers. Most of the recently developed robots are already equipped with small computers running an operating systems capable of clustering. Only some software packages (named middleware) and applications should be installed on the robots to form a robotic cluster. This paper will describe the details of these packages in section II-D and III.

C. *Programming*

Programming for robotic clusters is different from other types of robotic programming. Programs should be written using parallel programming approaches. The programmer should consider the following questions:

- Which parts of the code should be run simultaneously in more than one robot and which parts should be run only on one robot?
- Which message passing protocol is better for a specific application?
- Which variables should be local in a robot and which variables should be global to all robots?
- How and when should the robots communicate?
- How should the local results of individual robots be aggregated to find the global result?

The answers of these questions depend on the cluster's architecture and the robotic application. For example, programming on a cluster with shared memory for the whole system is very different from a cluster that does not have any shared memory. Since in many robotic systems, the robots are structurally independent, the programmer should not use any shared memory approach and should try to minimize the number of messages to be sent and received. Based on the application, the programmer should design an algorithm and address the mentioned questions. Section IV particularly provides an example and presents a parallel algorithm for a robotic application. Generally speaking, most of the parallel programs are written in C language using message passing protocols provided by clustering middleware for communication between the nodes. Section II-D presents details of these protocols.

D. Middleware

In a robotic cluster (similar to a computer cluster), the activities of the computing nodes (robots) are orchestrated by "clustering middleware", a software layer that allows the users or applications to treat the cluster as one cohesive computing unit, e.g. via a single system image (SSI) concept [48]. Middleware actually resides between the operating system and the user applications. It glues hardware resources by message passing, moving processes across machines, monitoring and synchronizing work of nodes [49]. Resource management, advanced task scheduling, workload management, communications, tools for cluster management, and providing cluster building kits are the most important issues in clustering that middleware standards provide solutions for them.

There are many different implemented standards for the middleware in computer clusters. Among these standards, three of them are well accepted by the community; Open Multi-Processing (OpenMP) [50], Parallel Virtual Machine (PVM) [51], and Message Passing Interface (MPI) [52]. OpenMP can only be run in shared memory computers, being not appropriate for robotic clusters. In contrast, PVM is a distributed operating system that forms a framework for building a single high performance parallel virtual machine from a collection of interconnected heterogeneous computers [51]. Since PVM combines the nodes and generates a unique virtual machine, it is not very appropriate for the distributed robotic systems that need to be individually independent. MPI software are widely accepted standards for communication among nodes that run a parallel program on a distributed-memory system. MPI defines an interface for a set of functions and libraries that can be used to pass messages between processes on the same computer or on different computers. MPI libraries offer easy ways of parallel programming techniques for memory distributed systems (as well as shared memory systems). Using these libraries, programmers can define which parts of the code run on the cluster and which parts on local node. Moreover, the programmer can declare which variables are globally passed to the other nodes of the robotic cluster and which variables are local. Therefore, MPI standard is the most appropriate protocol for robotic clusters and provides all of their necessary requirements. Section III presents the exact MPI middleware that is used in this study.

E. Load distribution

Load distribution is the action of partitioning the total task to several subtasks which can be assigned to different processing robots to be run in parallel. When these subtasks are assigned to the nodes (robots), they are called load. Load distribution is a very important problem specially when the robots are heterogeneous and the application is unbalanced. Robots submit jobs to the cluster at random times. When application is well balanced (similar to the case study of this paper in section IV), pure MPI programs usually result in good application performance [53]. The problem appears when application has internal unbalance load. If the load unbalance is static, analyzing the application and performing the distribution accordingly can solve the problem [54]. However, if load unbalance is dynamic, more complex methodologies should be used to increase the efficiency of the system. Several researchers have addressed the problem of dynamic load balancing in computer clusters. In [53] the proposed system dynamically measures the percentage of computational load from the different MPI processes and, according to that, it redistributes processes among them. Bhandarkar et al. [55] presented an implementation of Adaptive MPI (AMPI) that supports dynamic load balancing for MPI applications. Utrera et al. [56] proposed a mechanism, the Load Balancing Detector (LDB), to classify applications dynamically, without any previous knowledge, depending on their balance degree and apply the appropriate process queue type to each job. Erciyes and Payli [57] developed a framework and a protocol to perform dynamic load balancing in grids of heterogeneous computational resources. Zhu et al. [58] presented a study on dynamic load balancing based on MPICH[4] libraries.

In general, there are two key parameters for dynamic load balancing in a robotic cluster:

1) the middleware of a cluster that should provide required message passing protocols so that the robots will be able to know and estimate the other robots processing status,
2) a robotic algorithm that has to be able to dynamically partition the total task to produce fine granulated subtasks which can be efficiently distributed between the robots.

Since the first parameter is available in any robotic cluster, it is the duty of the programmer to address the latter issue in the design of the robotic algorithms. In robotic clusters (similar to computer clusters), depending on the specific application, the system designer should choose a dynamic/static load balancing method. Section IV-B1 goes to the details of a real example of load distribution in a real robotic cluster application.

F. Communication

Communication is a fundamental key in clustering. For the physical layer of communication of a robotic cluster, any standard wireless architecture can be used. However, in most wireless LAN architectures, routers play a significant role in the distribution of the robots in the space. To get into the cluster, robots should physically approach to a router. On the other hand, using an ad-hoc architecture (e.g. OLSR[5] or BATMAN[6]) that the robots connect to each other and establish a network wirelessly and automatically without using a router, the relative spatial topology of the robots is an important point

[4]http://www.mcs.anl.gov/research/projects/mpi/mpich1
[5]www.olsr.org
[6]http://www.open-mesh.org

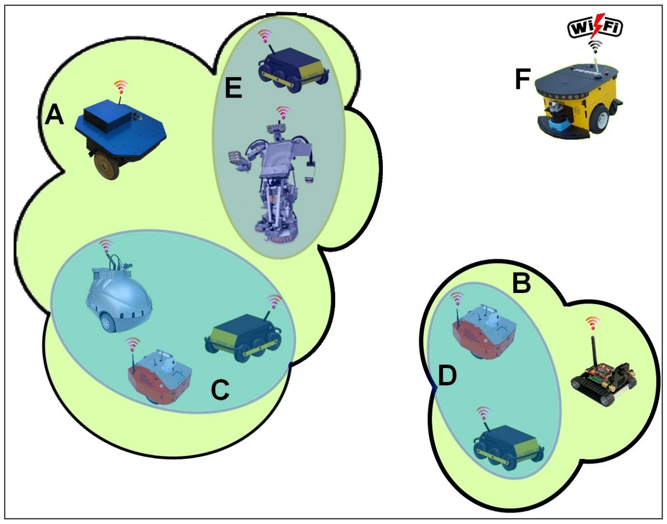
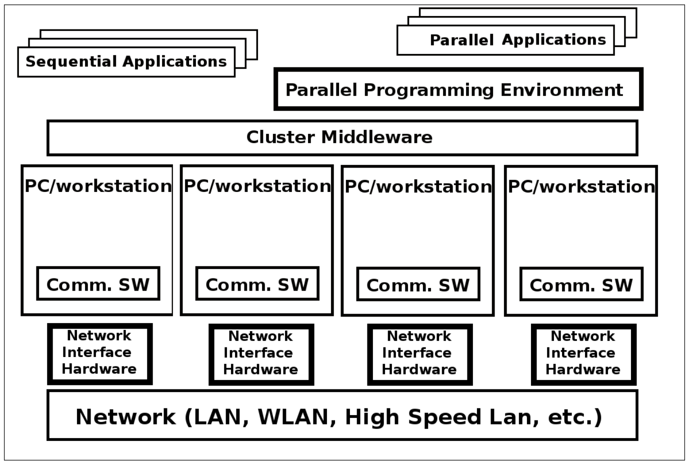


Fig. 2. Dynamic mobile robotic clusters. The robots have established 2 networks and 3 computing clusters. A: Coverage area of a wireless ad-hoc network established between six robots. B: The coverage area of the second network. C: Three robots inside network A have generated a cluster to share their processing resources. D: Two robots in network B have established a cluster. E: Two robots in network A have established another cluster. F: a robot that is not inside any network or cluster.



Sequential Applications
Parallel Applications
Parallel Programming Environment
Cluster Middleware
PC/workstation
PC/workstation
PC/workstation
PC/workstation
Comm. SW
Comm. SW
Comm. SW
Comm. SW
Network Interface Hardware
Network Interface Hardware
Network Interface Hardware
Network Interface Hardware
Network (LAN, WLAN, High Speed Lan, etc.)

Fig. 3. Computer cluster architecture

in the performance and functionality of the cluster. In case of existence of routers, the wireless routers can be carried by the robots, thus the network zone changes while the robots which carry the routers (called *router robots*) move. Fig .2 illustrates this characteristic of robotic clusters. The position of the router robots relative to the other robots leads to different configurations for the robotic cluster. This issue arises several questions that need to be answered in this field:

- How should the group of robots move in the environment in order to get more advantages from the clustering?
- What are the differences in behaviors of the router robots and the other robots?
- How far can a robot get from the mobile routers in different conditions?
- What should be the spatial formation of the router robots considering the position of the other robots and the environment to keep the cluster established?
- Given a fixed configuration, which robots should establish a cluster to increase the efficiency of the group?

Similar questions hold in the case of an ad-hoc robotic cluster where there is no router. It is obvious that these questions do not have unique answers in different applications/conditions and should be studied by the community in any application that this concept will be used.

Despite most of the works in computer clusters that start with the assumption of perfect communication, namely no delay and unlimited bandwidth [59], robotic clusters consist of loosely coupled distributed robots that simultaneously execute multiple tasks and imply different and unpredicted delays in cluster parallel processing. There are several works on studying [60], [61], modeling [59], and dealing with the communication issues (namely latency, overhead and bandwidth) in parallel processing with computer clusters that can be adapted to robotic clusters as well.

The impact of communication overhead on the system performance is highly dependent on the application [60]. In a robotic cluster, communication occurs when a robot dispatches a task to the robots in the cluster and the robots return computed results. Performance degradation occurs when communication overhead is relatively large compared with execution time of a single task. Therefore, the robotic cluster concept is more useful in the applications that the amount of data that should be exchanged between the robots is low while the required processing is huge; thus the impact of communication overhead on performance is insignificant.

Due to dynamic distribution of the robots in the environment and their loose communication connection, one of the main problems in a robotic cluster is how to deal with communication failures. Cluster's middleware provides libraries and functions that a user or an application can check and find the available working nodes in the cluster. The middleware layers in different nodes send and receive especial messages to each other and they are always aware of the others nodes' status. On the other hand, the algorithms in robotic clusters should consider this problem. The algorithm should be designed such that if one or more nodes in the cluster fail or lose the communication connection before or after load distribution, the system still be able to manage the situation and solve the problem. Section IV-B1 presents this issue in a particular application in details.

G. Dynamic architecture

Despite computer clusters that their hardware structure is either fixed or changed by a human administrator, in robotic clusters the hardware structure of the system dynamically changes while the robots spatially move in the environment. The robots eventually (and sometimes intentionally) get in, or get out of, the network range, making the hardware structure of the cluster very dynamic. This is one of the most significant

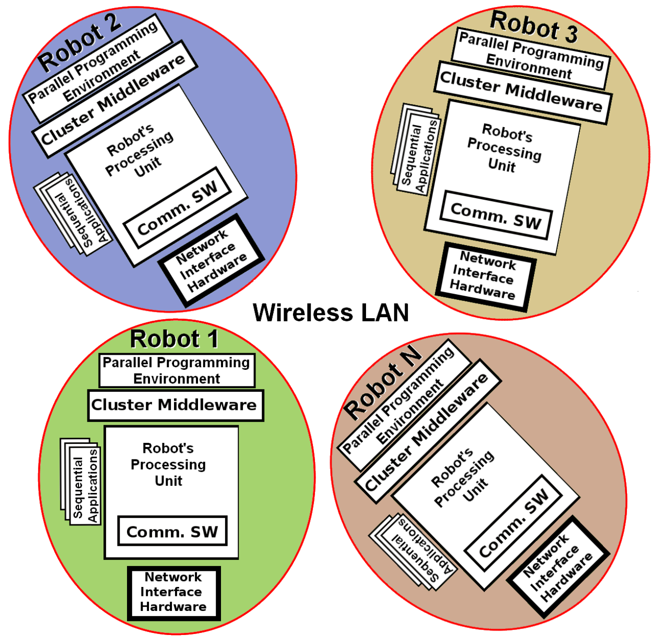


Fig. 4. Robotic cluster architecture

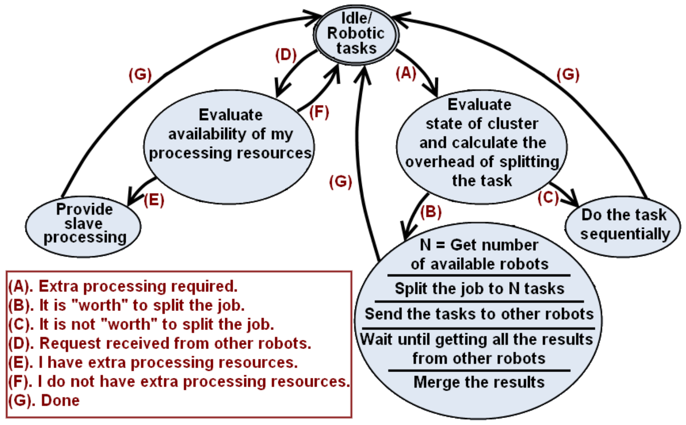

Idle/ Robotic tasks
Evaluate availability of my processing resources
Evaluate state of cluster and calculate the overhead of splitting the task
Provide slave processing
Do the task sequentially
N = Get number of available robots
Split the job to N tasks
Send the tasks to other robots
Wait until getting all the results from other robots
Merge the results
(A) (B) (C) (D) (E) (F) (G)
(A). Extra processing required.
(B). It is "worth" to split the job.
(C). It is not "worth" to split the job.
(D). Request received from other robots.
(E). I have extra processing resources.
(F). I do not have extra processing resources.
(G). Done

Fig. 5. Clustering and task distribution for individual robots.

differences between a computer cluster and a robotic cluster.

A typical architecture of a computer cluster is shown in Fig. 3. The key components of a cluster include multiple standalone computers, operating systems, a high performance interconnection, communication software, middleware, and applications. Fig. 4 illustrates the architecture of a robotic cluster. In this architecture, each robot is an independent standalone node that has minimum requirements and is free to join to a cluster or disconnect from the cluster.

To establish robotic clusters, the robots should automatically decide to connect or to disconnect from the neighboring robots. This issue raises the questions of when and how the robots can make these decisions. The answer to these questions depends on many parameters including:

- The robotic application: Not all the robotic problems that need high processing resources can take advantage of clustering; in some applications the nature of the problem is sequential and it cannot be divided to smaller subproblems to be solved by multiple processing nodes. This issue has been already studied deeply in parallel programming literature [62].
- The amount of required and available processing resources: The robots which lack processing resources (called *requesters*) or have extra processing resources (called *providers*), should join together to establish a cluster and help each other. The role of the robots as being *requesters* or *providers* can dynamically change in a robotic cluster.
- The overhead of the clustering and the speed of the network: The robots which request for using processing resources of the cluster should measure the overhead of clustering for that special task before sending the request. The overhead of clustering depends on many parameters including network's speed, the amount of data that needs to be exchanged between the robots, the speed of slowest node of the cluster and the level of parallelism of the robotic application.

Fig. 5 presents a state diagram that individual robots in a robotic cluster follow in their decision makings. In this diagram, the robots which do not need and have extra processing resources broadcast a message of "I am available" to the neighboring robots. On the other hand, the robots which lack processing resources, measure the current available processing resources in the cluster and the overhead of using the network and check if it is *worth* to split the task or not. We define the following conditions as the meaning of being "worth":

$$T_p + T_{overhead} < T_{seq} \qquad (1)$$

Where, T_p is the estimated time that it takes for the nodes of the cluster to finish processing their share of task. T_p is mostly relative to the time that the computationally slowest robot of the cluster takes to process its subtask. T_{seq} is the estimated time that it takes for the robot to sequentially process its own task. $T_{overhead}$ is defined by following formula:

$$T_{overhead} = T_{split} + T_{comm} + T_{Agg} \qquad (2)$$

T_{split} is the estimated required time for splitting the task, T_{comm} is the estimated communications time and T_{agg} is the time required for aggregating the distributed results and provide the final result of the process.

If condition (1) holds, the robot splits its task into N subtasks and sends requests to the available robots to process the subtasks. N is the number of current available *providers* in the cluster. The robot waits until getting all results from the other robots and then it merges the results. On the other hand, if condition (1) does not hold, the robot performs the task sequentially on its own processor.

H. Spatial distribution

In spite of computer clusters, in robotic clusters, nodes are distributed in the space and they are able to interact with the

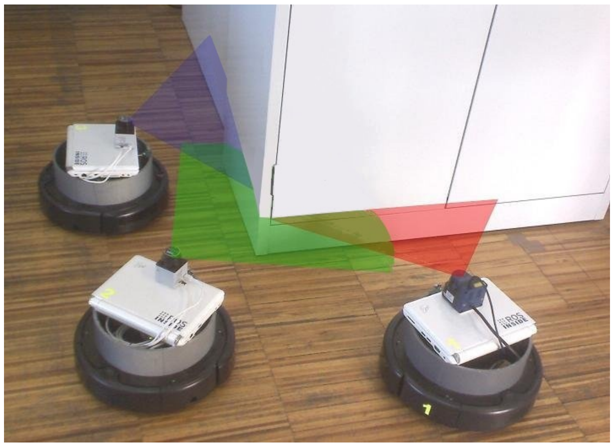


Fig. 6. Spatial distribution of robotic clusters, a conceptual picture.

environment by their sensors and actuators. In some robotic applications this point may provide several profits/interests. For instance, environmental features detection/extraction, that is one of the main challenges in many robotics applications namely search, exploration, patrolling, coverage, mapping, etc, can benefit from this characteristic of robotic clusters. Fig. 6 illustrates several robots that are distributed in the area and extract the environmental features using the clustering concept. The robots can sense the environment from different points of view and share their acquired data with each other through the cluster. By running a parallel algorithm in this cluster the robots can cooperatively process the shared data and extract/detect the feature. Therefore, in addition to higher processing capabilities, the spatial distribution of the robotic clusters might be a beneficial point in some applications. These concepts need to be more studied by the community in future.

Most of the above mentioned issues related to the robotic clusters are dependent on the application. To present the details of this concept and to evaluate its performance, a case study is demonstrated in section IV as an example of robotic applications.

III. Implementation of a robotic cluster

We have designed and implemented a robotic cluster using modified iRobot Roomba[7] robots. The iRobot Roomba robot is an attractive platform because it is inexpensive, readily available and can be fully monitored and commanded through a serial port interface. In this work, a set of eight Roomba robots were upgraded with small laptop computers (ASUS Eee PC 901 and ASUS Eee PC 1015PEM) running Ubuntu[8] operating system and the Player[9] environment to control the robots. Fig. 7 shows the implemented robotic cluster.

After configuring some necessary parameters (e.g. IP addresses) on the robots, an MPI software is needed to be installed. MPI is actually a library of routines that can be called usually from Fortran or C programs. There are a large number of implementations of MPI, two open source versions are LAM[10] and MPICH. LAM is an effective way for fast client-to-client communication and is portable to all Unix-based machines. MPICH is another freely available, portable implementation of MPI, a standard for message passing for distributed memory applications used in parallel computing, available for most flavors of Unix and other operating systems.

[7]http://www.irobot.com
[8]http://www.ubuntu.com
[9]http://playerstage.sourceforge.net

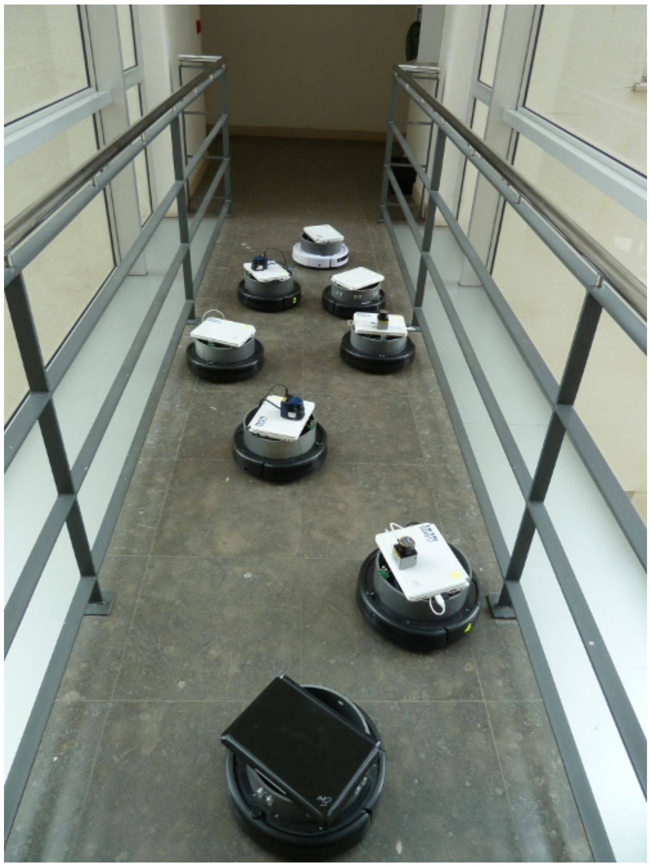


Fig. 7. Implemented robotic cluster

We installed MPICH libraries because it provides more programming facilities on Ubuntu and its installation is straight forward. After installing LAM or MPICH or a similar MPI library, the cluster is established, it is only needed to write and compile parallel programs on the robots. Compiling MPI programs can be done with several compilers, e.g. *mpic++*, with extended MPI libraries.

IV. Case study: map merging in a robotic cluster

As stated in section II, the concept of robotic clusters is useful when the robots need high processing capability in short periods of their operations. There are many robotics problems with these characteristics. One of these problems is "topological map merging". This section goes to the details of this problem and provides a solution based on the robotic clusters.

[10]http://www.lam-mpi.org

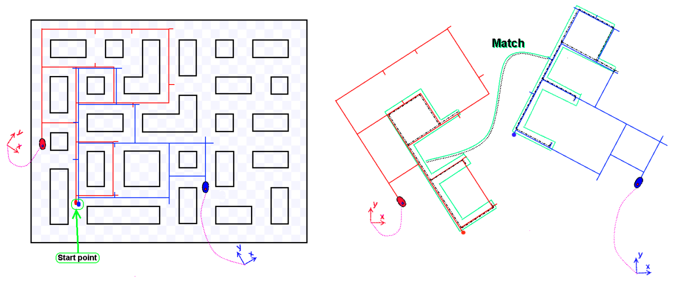


Fig. 8. An environment being explored by two robots with different coordinate systems (left). Matching generated maps of two robots exploring the environment (right).

Mapping with multiple robots has received much attention in the recent years due to its variety of applications such as planning. Consider several robots navigating in an environment and each one has its own coordinate system. Their X and Y axes do not match with each other and even they do not know where the reference point of the other's localization system is. Each robot is generating its own topological map of visited local area. Topological maps provide a brief characterization of the navigability of a structured environment, and, with measurements collected during exploration, the vertexes of the map can be embedded in a metric space [6]. These maps use a graph to represent possibilities for navigation through an environment and need less memory than their metric counterparts. The robots repeatedly send these local self generated topological maps to each other. The submaps generated by individual robots contain translation and orientation errors. The problem is how each one of them can integrate the data coming from the others to its local map and generate a more complete map. Fig. 8 shows an example of this problem.

While most research on multi-robot mapping has addressed the problem by creation of occupancy grid maps ([63], [64], [65]), some research has been done on feature based or topological maps ([3], [8], [66], [67], [68]). Jennings et al. [69] used individual robots to create topological partial maps, and then used a simple distance metric to merge the maps considering a global reference frame for all robots. Dudek et al. [70] created maps of a graph-like world under the assumption that all robots start from the same point in the graph. Konolige et al. [71] analyzed and showed the efficiency of taking a feature-based approach to merging instead of attempting to match occupancy data.

Most of the works in multi-robot mapping assume that all robots in the system have a common reference frame, an assumption not made in this paper. A few notable exceptions tackle the problem without a common reference frame. Ko et al. [65] presented a method in which robots exchange occupancy maps and localize themselves in each others maps using particle filters. Dedeoglu and Sukhatme [66] proposed a different approach for merging landmark-based maps without a common reference frame. They used heuristics to estimate a transformation between two maps using a single-vertex match found and paired other vertexes that are approximately close to each other under this transformation. The work most related to ours is that of Huang and Beevers [6] who have presented a method that creates vertex and edge pairings using the structure of the maps and then estimates a transformation using this match. By comparing the map structure, mismatches can be discard earlier in the algorithm, and the transformations can be computed using multiple-vertex matches instead of single-vertex matches. In our previous experiments in [8], it was assumed that although there is no common reference frame, the robots start the mission from the same initial node. This assumption made it easier to solve the problem of map merging with multiple robots. Here we do not make any of these assumptions and present an approach for this problem considering no common reference frame and no common initial node. Moreover we develop a novel parallel algorithm for this problem to run in robotic clusters using parallel programming approaches.[11]

Usually, in exploration and mapping scenarios the robots navigate in the environment and whenever they get into a new environmental feature they send the data of the updated map to the other robots. In certain conditions each robot will integrate its own achieved local maps with the other robot's maps. This task is computationally intensive since it requires an exhaustive search of data. In this case the robot needs a high processing resource to analyze the maps and merge them together.

A. *Sequential solution*

Algorithm 1, *Merge_seq*, works on two given maps $M_1 = [V,E,k_1]$ (where V is the set of vertexes V_i , E is the set of edges e_i and k_1 (called size) is the number of edges of the map) and $M_2 = [W,F,k_2]$ and finds the largest common connected subgraph between these two and calculates the best transformation function that maps M_1 on M_2. This transformation function matches the coordinate reference frames of the two maps. Therefore, the final merged map is derived by overlapping only the vertexes' positions using the transformation function.

This paper defines the stated terms in the algorithms as following:

- "$V\_equiv(v,w,F)$": checks if vertexes v and w are equivalent. Two vertexes are equivalent if their environmental features match with each other and their positions can be approximately matched by function F.
- "$E\_equiv(e_i, f_j)$": Two edges e_i and f_j are equivalent if their lengths are approximately equal, i.e. $Length(f_j) - \Delta_T < Length(e_i) < Length(f_j) + \Delta_T$ and their connecting vertexes are equivalent. Δ_T is a distance threshold that should be defined based on the model of robots' localization error.
- "Transform_func(V_i,W_j,V_k,W_l)": calculates a transformation function that maps vertex V_i on W_j and vertex V_k on W_l. Having the positions of two vertexes in map M_1 and two vertexes in map M_2, with a geometric calculation

[11]The source code of all of the presented algorithms (serial and parallel) is available online in http://www.isr.uc.pt/∼ali/mapmerging

Algorithm 1: Merge_seq(M_1,M_2,MAP)

```
inputs:
        $M_1 = [V, E, k_1]$
        $M_2 = [W, F, k_2]$
Output: A map that is the result of merging $M_1$ and $M_2$
begin
    LIST = Null
    for $i = 1; i < k_1; i++$ do
        for $j = 1; j < k_2; j++$ do
            if $E\_equiv(e_i, f_j)$ then
                $(I_1M_1, I_2M_1)$ = Index(Adj($e_i$))
                $(I_1M_2, I_2M_2)$ = Index(Adj($f_j$))
                $F$ = Transform_func($V_{I_1M_1}, W_{I_1M_2}, V_{I_2M_1}, W_{I_2M_2}$)
                L_equ_v = Null
                $S$ = Com_subgraph($M_1, M_2, F, I_1M_1, I_1M_2, 0$)
                if $S > 0$ then
                    add [$F, S, L\_equ\_v$] to $LIST$
                $F$ = Transform_func($V_{I_2M_1}, W_{I_1M_2}, V_{I_1M_1}, W_{I_2M_2}$)
                L_equ_v = Null
                $S$ = Com_subgraph($M_1, M_2, F, I_2M_1, I_1M_2, 0$)
                if $S > 0$ then
                    add [$F, S, L\_equ\_v$] to $LIST$
    $I$ = the index of largest S saved in $LIST$
    if $I$ == $Null$ then
        return(Null)
    $F_I$ = Transformation function saved in $LIST(I)$
    $L\_equ\_V_I$ = List of equivalent vertexes saved in $LIST(I)$
    $M1_{mapped} = F_I(M_1)$
    MAP = Save_overlap($M1_{mapped}, M_2, L\_equ\_V_I$)
    return(MAP)
end
```

Algorithm 2: Com_subgraph($M_1, M_2, F, IG1, IG2, size$)

```
inputs:
        $M_1 = [V, E, k_1]$, $M_2 = [W, F, k_2]$
        $F$ = The transformation function that maps $M_1$ on $M_2$
        $IG1$ = the index of the current under process vertex of $M_1$
        $IG2$ = the index of the current under process vertex of $M_2$
        size = the current size of common subgraph
Output: Size of the largest common subgraph
begin
    if $!(V\_equiv(V_{IG1}, W_{IG2}, F)$ then
        return 0
    if $[IG1, IG2] \notin L\_equ\_v$ then
        L_equ_v = L_equ_v $\cup$ {[$IG1, IG2$]}
        size ++
        for $N$ = 1; $N$ < Max Node Deg ; $N$++ do
            $IG1 = Link(V_{IG1}, M_1, F, N)$
            $IG2 = Link(W_{IG2}, M_2, 1, N)$
            if $\exists\ V_{IG1}$ & $\exists\ W_{IG2}$ then
                size = Com_subgraph($M_1, M_2, F, IG1, IG2, size$)
            if $size$ == $0$ then
                return 0
    return(size)
end
```

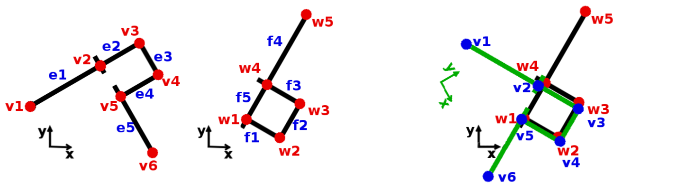


Fig. 9. Two maps with different coordinate systems (left). The result of merging maps (right).

a transformation function can be simply extracted that contains translation and rotation parameters.

- "$Adj(e_i)$": Returns the two vertexes of edge e_i.
- "Index(Adj(e_i))": Returns the indexes of two vertexes connected to edge e_i.
- "$Link(V_{IG1}, M, F, N)$": Finds the Nth connected vertex to V_{IG1} in map M, considering the rotation parameter in function F.
- "Max Node Deg" is the maximum degree of vertexes in the maps, i.e. the maximum number of edges connected to a vertex.

Algorithm *Merge_seq* compares all the edges in M_1 and M_2 one by one. Once an edge e_i in M_1 is found equivalent of an edge f_j in M_2, a transformation function F will be calculated that maps e_i on f_j. Then the algorithm 2, *Com_subgraph*, finds the largest common connected subgraph between the two maps starting from the edge e_i and f_j to check the compatibility of the vertexes based on the transformation function F. Since each edge has two vertexes, for every common edge, the algorithm 2 is applied two times; once assuming that the first vertex of e_i is matched with the first vertex of f_j and once assuming the second vertex of e_i is matched with the first vertex of f_j.

Com_subgraph is a recursive algorithm that creates lists of corresponding vertexes between the two maps (each list is called a *hypothesis*) by locally expanding single vertex matches. It starts with the starting vertexes, compares them, and if they are compatible it finds the next connected vertex to these vertexes and runs the algorithm with the new found vertexes. If the algorithm finds any conflict (mismatch), it returns zero indicating that matching e_i and f_j using function F will lead to a conflict in the rest of map edges and this match should be ignored. Otherwise, if there were no mismatch during the process of *Com_subgraph* algorithm, it saves the list of compatible vertexes in *L_equ_v* and returns the number of the found common connected subgraph (one hypothesis). Algorithm *Merge_seq* saves the data (size, matched vertexes and the transformation function) of the generates hypothesis in *LIST* and then processes the next edges in M_1 and M_2. Finally, the *LIST* will contain all common subgraphs that can be generated between M_1 and M_2 with different transformation functions. Algorithm *Merge_seq* finds the biggest hypothesis in the *LIST* and generates $M1_{mapped}$ based on its transformation function. Eventually, $M1_{mapped}$ and M_2 have common coordinate reference frames and we have the list of matched vertexes between these two, merging these two maps can be easily done by overlapping them on each other and generate a new map.

Table I shows how the algorithm *Merge_seq* and *Com_subgraph* work on two example maps of Fig. 9.(left). Starting from e_1 and f_1, equivalent edges are $(e_1 - f_4)$, $(e_2 - f_1)$, $(e_2 - f_2)$, etc. These edges are listed in the first

TABLE I
LIST OF SIMILAR EDGES AND PROCESSED VERTEXES IN ALGORITHMS 1 AND 2 FOR THE MAPS SHOWN IN FIG. 9.(LEFT). MISMATCHES ARE COLORED RED IN THIS TABLE.

Edges	Processed vertexes	Edges	Processed vertexes
$e_1 - f_4$	V_1W_4	$e_3 - f_2$	V_3W_2, V_2W_1
$e_1 - f_4$	V_1W_5, V_2W_4, V_3W_1	$e_3 - f_2$	$V_3W_3, V_4W_2, V_5W_1, V_2W_4$
$e_2 - f_1$	V_2W_1	$e_4 - f_1$	$V_4W_2, V_5W_1, V_3W_3, V_2W_4$
$e_2 - f_1$	V_3W_1	$e_4 - f_1$	V_4W_1
$e_2 - f_2$	V_2W_2	$e_4 - f_2$	V_4W_2, V_5W_3
$e_2 - f_2$	V_3W_2, V_4W_1	$e_4 - f_2$	$V_4W_3V_5W_2$
$e_2 - f_3$	V_2W_3	$e_4 - f_3$	V_4W_3, V_5W_4
$e_2 - f_3$	$V_3W_3, V_4W_2, V_5W_1, V_2W_4$	$e_4 - f_3$	V_4W_4
$e_2 - f_5$	V_2W_1	$e_4 - f_5$	V_4W_1
$e_2 - f_5$	V_3W_1	$e_4 - f_5$	V_4W_4
$e_3 - f_1$	V_3W_1	$e_5 - f_4$	V_5W_4
$e_3 - f_1$	V_3W_2, V_4W_1	$e_5 - f_4$	V_5W_5

column of table I. Algorithm *Com_subgraph* compares the vertexes of the two maps, starting from the ones that are connected to the equivalent edges, that generates hypotheses. The second column of table I has listed the vertexes that are compared for each pair of edges. The mismatches are colored red. For instance, processing equivalent edges ($e_1 - f_4$), algorithm *Com_subgraph* considers the first hypothesis which includes V_1,W_4. Since V_1 is of degree one and W_4 is of degree four, they do not match, so the algorithm does not continue processing this hypothesis and returns zero (V_1W_4 is colored red in the table to indicate the mismatch). Processing current equivalent edges ($e_1 - f_4$), the algorithm takes next hypothesis into account which starts with V_1,W_5. These vertexes (V_1 and W_5) are compatible so the next vertexes V_2 and W_4 will be compared. Since these two are also compatible the next vertexes V_3 and W_1 are compared that are incompatible (so we have V_1W_5, V_2W_4, V_3W_1 in the table). Finally algorithm 1 will have the list of all hypotheses saved in *LIST*. For the mentioned example above, *LIST* contains three hypotheses because only those which do not lead to any mismatch are saved in *LIST*. These hypotheses are colored blue in table I. Based on the biggest found hypothesis (or one of the biggest ones), algorithm *Merge_seq* will map the M_1 with the associated transformation function to that hypothesis, this action will overlap M_1 on M_2 (Fig. 9.(right) shows this overlap). Finally the complete map is driven by overlapping matched submaps.

B. Parallel solution

We provide a parallel solution for the topological map merging problem using message passing interface (MPI) standard. This is the first parallel implementation of topological map merging problem. Algorithm 3 demonstrates the pseudocode of the proposed algorithm for parallel topological map merging. The parallel part of this algorithm, lines 8 to 30, is run on multiple robots of the cluster and the rest of the code is run serially only on the robot which has initially started the program.

1) Load distribution: Following the flowchart in Fig. 5 each robot individually makes its decision to connect or to establish a cluster or exit from it. When a robot needs to distribute a task between the other robots in the cluster, it first should know how many robots have joined to the cluster and then distribute the load between them. Section II-E mentioned the two key parameters of load balancing in robotic clusters i.e. middleware and parallel algorithms.

Algorithm 3: Merge_parallel(M_1,M_2,MAP), MPI solution

```
1  inputs:
2          $M_1 = [V, E, k_1]$
3          $M_2 = [W, F, k_2]$
4  Output: A map that is the result of merging $M_1$ and $M_2$
5  begin
6      MPI_Init(...)
7      MPI_Barrier(MPI_COMM_WORLD)
8      begin
9          MPI_comm_rank (MPI_COMM_WORLD, &id)
10         MPI_comm_size (MPI_COMM_WORLD, &p)
11         $Low\_value = (id - 1) \times k_1/p + 1$
12         $High\_value = (id) \times k_1/p$
13         LIST = Null
14         for $i = Low\_value; i < High\_value; i++$ do
15             for $j = 1; j < k_2; j++$ do
16                 if $E\_equiv(e_i, f_j)$ then
17                     $(I_1M_1, I_2M_1)$ = Index(Adj($e_i$))
18                     $(I_1M_2, I_2M_2)$ = Index(Adj($f_j$))
19                     $F$ = Transform_func($V_{I_1M_1}, W_{I_1M_2}, V_{I_2M_1}, W_{I_2M_2}$)
20                     L_equ_v = Null
21                     $S$ = Com_subgraph($M_1, M_2, F, I_1M_1, I_1M_2, 0$)
22                     if $S > 0$ then
23                         add $[F, S, L\_equ\_v]$ to $LIST$
24                     $F$ = Transform_func($V_{I_2M_1}, W_{I_1M_2}, V_{I_1M_1}, W_{I_2M_2}$)
25                     L_equ_v = Null
26                     $S$ = Com_subgraph($M_1, M_2, F, I_2M_1, I_1M_2, 0$)
27                     if $S > 0$ then
28                         add $[F, S, L\_equ\_v]$ to $LIST$
29     end
30     MPI_Bcast (&LIST, ...)
31     if $id$ == $S_{id}$ then
32         MPI_Reduce (&LIST,...)
33         if !$MPI\_SUCCESS$ then
34             return($MPI\_ERROR$)
35         $I$ = the index of largest S saved in $LIST$
36         if $I$ == -1 then
37             return(NULL)
38         $F_I$ = Transformation function saved in $LIST(I)$
39         $L\_equ\_V_I$ = List of equivalent vertexes saved in $LIST(I)$
40         $M1_{mapped} = F_I(M_1)$
41         MAP = Save_overlap($M1_{mapped}, M_2, L\_equ\_V_I$)
42         return(MAP)
43 end
```

In terms of middleware, using the MPICH environment, the robots can use "*mpd*" commands (e.g., "*mpdboot*", "*mpdallexit*", etc.) to establish, connect to or exit from a cluster. Moreover, each robot is also able to find the number of available processing units in the robotic cluster by using the command "*mpdtrace*" or the function MPI_comm_size. For running a parallel program, the robots only need to run the command "$mpirun\ -np\ N\ APP$", where N indicates the number of robots that the program should be run on, and "*APP*" is the name of parallel program that should be executed (i.e. algorithm Merge_parallel in this case). A robot always can use the middleware libraries and commands to find the available working nodes and run the program exclusively on them.

In terms of parallel algorithm, the algorithm 3 dynamically distributes the load between the available robots in the cluster in real time. Algorithm 3 first finds the number of available operational nodes in the cluster (by MPI_comm_size in line 10) and then it distributes the load between them. Therefore, if one or more nodes in the cluster have failed or have lost the communication connection, algorithm 3 distributes the load only between the operational nodes. However, if after distributing the load, one of the robots fails (or loses the connection), the robot which has initially distributed the task will get a timeout (i.e. an MPI error message) instead of the results. The algorithm returns the error (lines 33 and 34 of algorithm 3) and this robot has to start over running the algorithm, so that it will distribute the load between the current available robots again.

The main source of potential parallelism in this application is data decomposition i.e. dividing problem search space between the robots. To automatically divide the search space between the robots (without having a central base station), this algorithm uses the robots' IDs as unique indicators. In algorithm 3 all of the processing nodes (robots) have complete data of two maps of M_1 and M_2 and each robot finds the biggest common subgraphs between the two maps that starts in its share of data. Each robot should work on a block (that is a division of whole search space) specified by two variables named Low_value and $High\_value$. Low_value_i and $High\_value_i$ are computed in robot i based on the following formulas:

$$Low\_value_i = (id_i - 1) \times \frac{k_1}{p} + 1 \tag{3}$$

$$High\_value_i = (id_i) \times \frac{k_1}{p} \tag{4}$$

where id_i denotes the ID of robot i in the cluster, p is the total number of nodes and k_1 is the size of map M_1. To explain these formulas, we provide an example; imagine the map size k_1 is 1000 and the *current* number of available robots p is 5. Based on (3) and (4), for the robot whose id is 1, Low_value_1 and $High\_value_1$ are respectively 1 and 200, and for the robot with id = 2, Low_value_2 is 201 and $High\_value_2$ is 400. Thus, Robot 1 works on interval [1,200], robot 2 on interval [201,400], robot 3 on [401,600] and so on. If the current number of robots was 10, the share of each robot would be intervals with length 100.

Using function MPI_comm_rank() in the algorithm the robots find their own ID and calling function MPI_comm_size() they will be aware of the total number of robots (p) participating in the cluster at the moment. It is worth to mention that in MPI programming, the variables are private, i.e. changes in a variable in a node (inside the parallel part of code) does not have any effect in the value of the same variable in another node.

By considering k_1 (and not k_2) in (3) and (4), map M_1 is divided between the robots in equal shares. Now each node

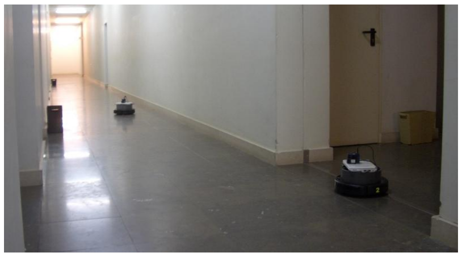


Fig. 10. Robots mapping a real environment [3]

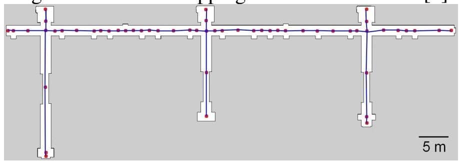


Fig. 11. The topological map with 54 nodes of the environment shown partially in Fig. 10.

runs the previously presented sequential algorithm but starting from its given share of data in M_1 and finds the biggest common subgraphs between M_1 and M_2 (see algorithm 3).

2) Functionality: Similar to algorithm 1, algorithm $Merge\_parallel$ compares all the edges in a block of M_1 and M_2 one by one. Once an edge e_i in M_1 is found equivalent with an edge f_j in M_2, a transformation function F will be calculated that maps e_i on f_j. Then the algorithm 2, $Com\_subgraph$, finds the largest common subgraph between the two maps starting from the edge e_i and f_j to check the compatibility of the vertexes based on the transformation function F. Each robot saves its found common subgraphs into its $LIST$. Finally, the $LIST$s will contain all common subgraphs that can be generated between M_1 and M_2 with different transformation functions. These $LIST$s are generated by different robots and need to be aggregated to one node and be processed.

3) Aggregating the results: MPI_Bcast is called in the algorithm, so that each robot will broadcast its results ($LIST$) to the cluster. In this algorithm, the robot with $id = S_{id}$ is responsible to aggregate the distributed results and calculate the final result. S_{id} denotes the id of the robot which initially requested the map merging process. This robot runs MPI_Reduce to gather the results of processed data from the other computers to its own $LIST$. Based on the biggest found common subgraph (hypothesis), algorithm $Merge\_parallel$ will map the M_1 with the associated transformation function to that hypothesis and generates $M1_{mapped}$. Finally the output map (MAP) is generated by overlapping $M1_{mapped}$ and M_2.

C. experimental results

The algorithms have been run on the implemented robotic cluster (Fig. 7, explained in section III) using up to eight modified Roomba robots. For evaluation of the method, all algorithms are tested ten times with different sets of maps. Several topological maps were experimentally generated by real robots using the results of our past experiments in [3], [32], [8], [72]. Fig. 10 shows two robots exploring and

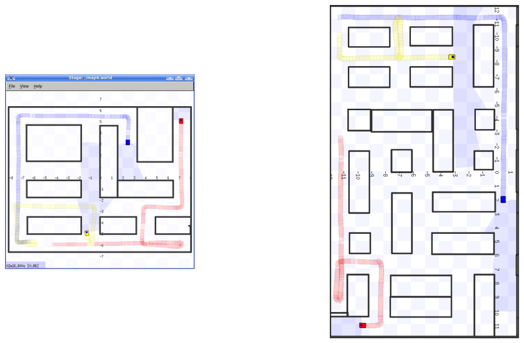


Fig. 12. Three robots mapping an environment with 36 nodes (left). Three robots mapping an environment with 81 nodes (right).

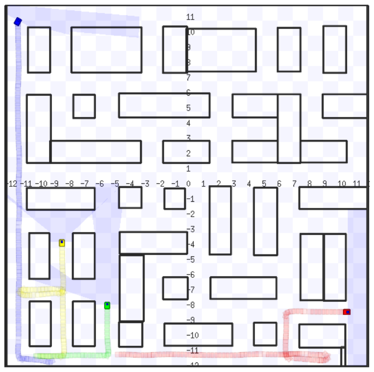


Fig. 13. Four robots mapping an environment with 136 nodes

mapping a real environment. This environment is the corridors of second floor of Institute of Systems and Robotics in Coimbra. Fig. 11 shows the topological map of this environment generated by the robots. Moreover, a set of exploration and mapping experiments were done in Player/Stage framework. Fig. 12 and Fig. 13 show three simulated environments being mapped by multiple robots.

Fig. 14 and 15 demonstrate an example of the map merging experiments. In Fig. 14 two partial maps of one environment are shown that were generated by two robots without having common reference frames. About 10 percent of vertexes and edges of these partial maps are equivalent. Fig. 15 presents the result of merging these maps. The execution time for each merging experience is measured five times and their mean values are shown in Fig. 16. These results show that if the maps are large enough (map size bigger than 54 in this case), execution time decreases as number of robots increase.

Parallel algorithms are usually evaluated by analyzing their speedup and their efficiency over the serial algorithms. Reduction of processing time by using more robots (speedup) can be calculated by:

$$Speedup = \frac{T_{seq}}{T_{Parallel}}$$

where:

- T_{seq}: execution time of code using one robot.
- $T_{parallel}$: execution time of parallel code using several robots

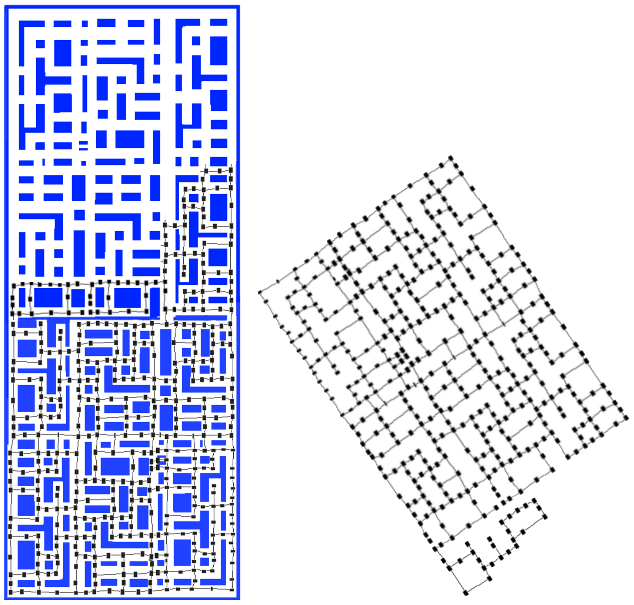


Fig. 14. Partial maps generated by 2 robots with different coordinate systems

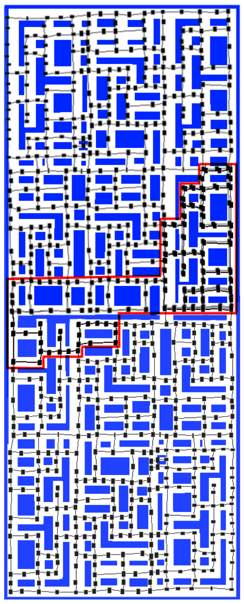


Fig. 15. The result of map merging algorithm for the maps of Fig. 14

A parallel algorithm is called “efficient” if its speedup is nearly linear to number of processors [73]. Fig. 17 presents the speedup of the parallel algorithm for different size of maps. It can be seen that speedup has almost linear relation to the number of robots for all size of N in large maps, thus, proving the efficiency of the proposed parallel algorithm. This means that having more robots in the cluster increases the speed of solving this problem. It should be mentioned that, according to the *Amdahl law* [74], [75], it is very difficult, even in an ideal parallel system, to obtain a speedup value equal with the number of processors because each program, in terms of

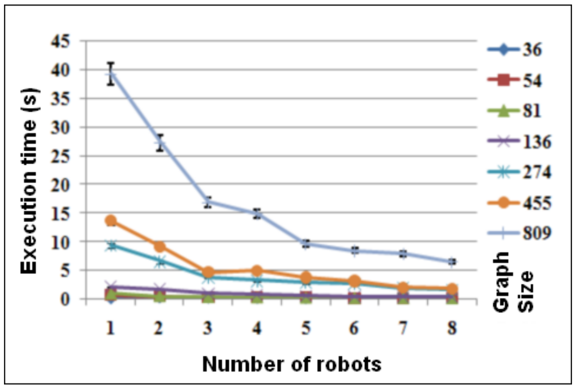


Fig. 16. Execution time results for different size of real world and simulation maps based on number of robots.

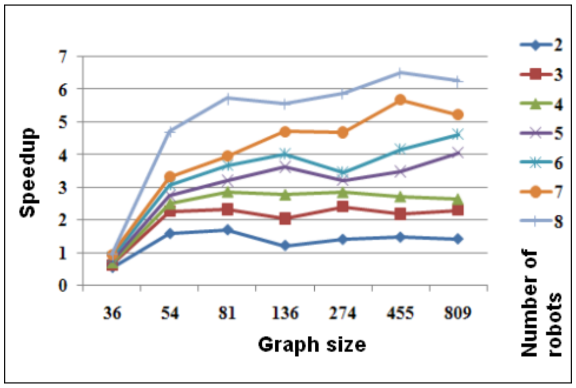


Fig. 17. Speedup for different sizes of problem in the real world and simulations.

running time, has a part that cannot be parallelized and has to be executed sequentially by one single processor.

The size, the topology and the degree of connectivity of the maps are also important effective parameters in the speed of the algorithms. Sometimes a smaller map takes more time to process than a bigger one because of the other parameters. For evaluating the methods in a more scientific way we generated several artificial maps similar to the maps of real structured environments by running a C program. The maps generated by this software are structurally similar to each other but their sizes are different. The maps are split to two submaps having about 10 percent of vertexes and edges in common. Then one of the submaps is rotated with a random angle and the results are fed to the algorithms as inputs. Fig. 18 shows two examples of these generated maps. Fig. 19 shows the merged map resulted from running the algorithms on the maps of Fig. 18. The execution time and speedup are measured for these experiments and are reported in Fig. 20 and Fig. 21. These results show that for large enough maps, the speedup increases by the number of robots in the cluster.

Another parameter to evaluate the performance of parallel algorithms is *efficiency* which measures the utilization rate of the processors in the execution of a parallel program. It is equal to the ratio of speedup and the number of processors used. We

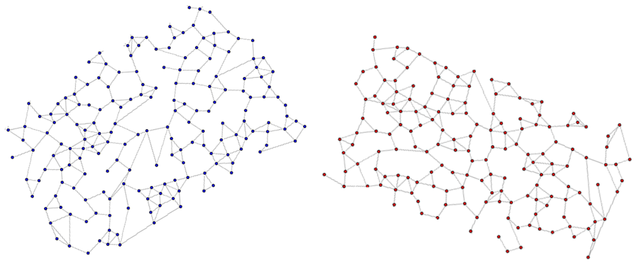


Fig. 18. Two given artificial maps to be merged by the algorithms.

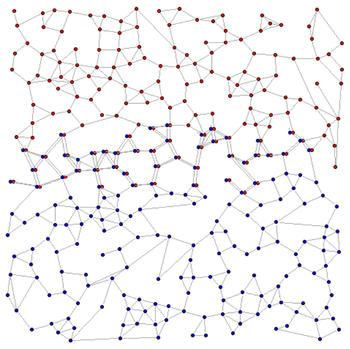


Fig. 19. Merged map with about 1000 vertexes and 900 edges.

Algorithm 4: Merge(M_1,M_2,MAP), with clustering solution

```
1  inputs:
2          $M_1 = [V, E, k_1]$
3          $M_2 = [W, F, k_2]$
4  Output: A map that is the result of merging $M_1$ and $M_2$
5  begin
6      if $(k_1 < M)$ and $(k_2 < M)$ then
7          Merge_seq($M_1$,$M_2$,MAP)
8          return(MAP)
9      else
10         Merge_parallel($M_1$,$M_2$,MAP)
11         return(MAP)
12 end
```

have measured this parameter for the proposed map merging parallel algorithm. Fig. 22 demonstrates the efficiency of the proposed parallel algorithm on the real world and simulation topological maps, while, Fig. 23 presents the efficiency of the algorithm on the artificially generated topological maps. Both graphs show similar results, i.e, the efficiency is mostly between 0.6 and 0.8 specially on large graphs it converges to 7.2. This is a significant result that proves the efficiency of the method in reality, since it was obtained by implementation of the algorithm on a real robotic cluster.

Considering both sequential and parallel solutions in algorithms 1 and 3, this question arises that “when should a robot use each of these algorithms?”. Considering the state diagram of Fig. 5, a robot should use the cluster only if it is “worth” to use it. The experimental results (in Fig. 17 , Fig. 21, Fig. 22 and Fig. 23) showed that if the size of maps is too small the parallel solution will take more time than the sequential solution. Based on this information, we present Algorithm 4

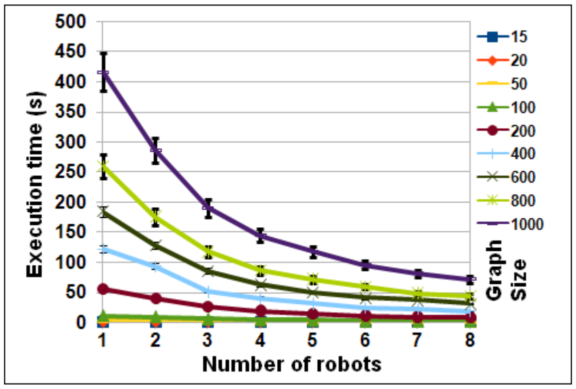


Fig. 20. Execution time results for different size of artificial maps based on number of processors

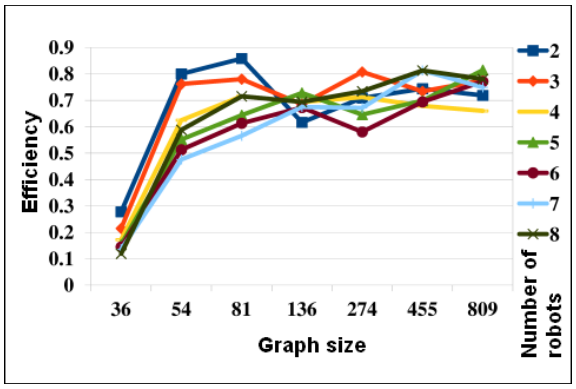


Fig. 22. The efficiency of parallel algorithm in different size of real and simulated topological maps.

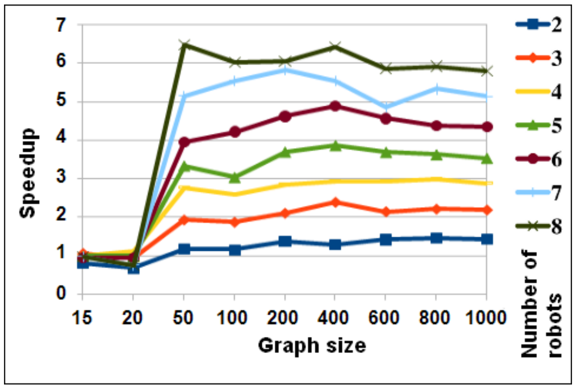


Fig. 21. Speedup for different sizes of artificial topological maps

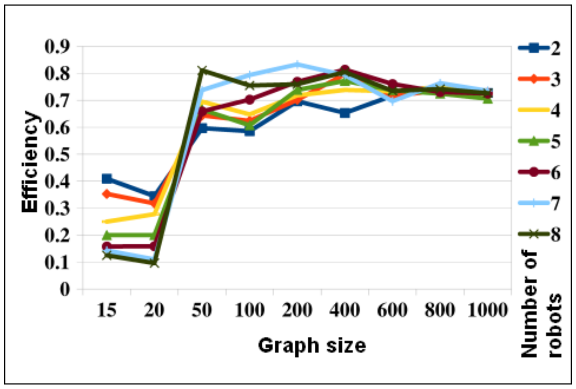


Fig. 23. The efficiency of parallel algorithm in different size of artificially generated topological maps.

that checks the size of input maps and calls the sequential algorithm if the maps are smaller than threshold *M* and calls the parallel algorithm if the maps are large enough.

For this particular application and the specified hardware and software, based on the speedup results of Fig. 17 and Fig. 21, we found that M should be set to 50. For other applications, the system designer should estimate or measure the system's performance in both sequential and parallel cases and design a method similar to algorithm 4 that uses the robotic cluster efficiently.

The presented results showed that the proposed method can speed up the execution time of a map merging algorithm. We believe that this approach can be used in many other multi-robot applications that demand for high processing resources.

V. Conclusions and discussions

This paper presented the novel concept of robotic clusters, an approach for better problem solving in distributed robotic systems. Using this approach, heterogeneous robots are able to share their processing resources in solving complex problems while they are still independent. The paper explained the characteristic, requirements and benefits of the method in multi-robot systems. A real world robotic cluster was implemented using eight robots and the problem of topological map merging was studied as a case study. The proposed parallel solution for topological map merging was another contribution of this paper. The method was tested with experimentally real-world, simulation, and artificially generated topological maps, and the results showed that robotic clusters increase the speed of robots in solving topological map merging problem. The idea of robotic clusters can be advantageous in many other multi-robot systems that (i) demand for high processing resources, (ii) require relatively cheap design, and (iii) intend to address problems that can be done in parallel with negligible overheads.

Finally, there are some issues that should be discussed here. An issue is the amount of data that should be exchanged between the robots in a cluster during the process. This highly depends on the application's nature and is part of clustering overhead explained in section II-G. In the presented case study in this paper, the size of very large topological maps is usually less than a few kilobytes and they are exchanged only at the beginning and at the end of the processing period. Considering the communication speed of wireless networks, the time required for transferring this amount of data is negligible in compare with the time required for the processing of this data. However, there are robotic applications that demand for high amount of data that need to be exchanged between the nodes if clustering is applied. This issue might increase the clustering overhead and thus decrease the speedup and efficiency of the clustering approach. This is an important parameter that the system designer should consider before designing a robotic cluster.

Another discussion can be done about the communication issues (namely latency, overhead and bandwidth). As section II-F already mentioned, this issues also are dependent on the application and are part of clustering overhead. For the particular case study of this paper and in the implemented robotic cluster, the communication overheads were insignificant for large graphs, however, for other applications, the system designer should estimate or measure the communication overheads to design a clustering method that is efficient and tolerant to possible failures.

Although this paper defined the concept of robotic clusters and presented its advantages, requirements, implementation, challenges and potential applications, still excessive research is needed to develop methodologies that allow designing and implementing robotic clusters to address several challenges in robotics research. Specially the spatial distribution of clustering robots in the environment and its effect on the system's performance should be studied in various applications in future. Finally, more dynamic clustering strategies should be developed by the community in order to optimize the usage of available processing resources in different conditions.

VI. Acknowledgment

The authors would like to thank the anonymous reviewers for their insightful and constructive comments that helped improving the manuscript. This work was partially supported by the Portuguese Foundation for Science and Technology contract SFRH/BD/45740/2008 and SFRH/BD/72697/2010.

References

[1] K. Leung, T. Barfoot, and H. Liu, "Decentralized localization of sparsely-communicating robot networks: a centralized-equivalent approach," *IEEE Trans. on Robotics*, vol. 26, no. 1, pp. 62–77, 2010.

[2] W. Sheng, Q. Yang, J. Tan, and N. Xi, "Distributed multi-robot coordination in area exploration," *Robotics and Autonomous Systems*, vol. 54, no. 12, pp. 945–955, 2006.

[3] A. Marjovi and L. Marques, "Multi-robot olfactory search in structured environments," *Robotics and Autonomous Systems*, vol. 52, pp. 867–881, 2011.

[4] A. Marjovi, J. Nunes, P. Sousa, R. Faria, and L. Marques, "An olfactory-based robot swarm navigation method," in *Proc. IEEE Int. Conf. on Robotics and Automation*, Alaska, USA, 2010.

[5] W. Burgard, M. Moors, C. Stachniss, and F. Schneider, "Coordinated multi-robot exploration," *IEEE Trans. on Robotics*, vol. 21, no. 3, pp. 376–386, 2005.

[6] W. Huang and K. Beevers, "Topological map merging," *The International Journal of Robotics Research*, vol. 24, no. 8, p. 601, 2005.

[7] D. Jung and A. Zelinsky, "Grounded symbolic communication between heterogeneous cooperating robots," *Autonomous Robots*, vol. 8, no. 3, pp. 269–292, 2000.

[8] A. Marjovi and L. Marques, "Multi-robot topological exploration using olfactory cues," in *Int. Symp. on Distributed Autonomous Robotics Systems*, Lausanne, Switzerland, 2010.

[9] C. Yu and R. Nagpal, "A Self-Adaptive Framework for Modular Robots in Dynamic Environment: Theory and Applications," *The International Journal of Robotics Research*, 2010.

[10] R. Beckers, O. Holland, and J. Deneubourg, "From local actions to global tasks: Stigmergy and collective robotics," in *Artificial life IV*, vol. 181. MIT Press, 1994, p. 189.

[11] R. Arkin, "Cooperation without communication: Multiagent schema-based robot navigation," *Journal of Robotic Systems*, vol. 9, no. 3, pp. 351–364, 1992.

[12] L. Steels, "A case study in the behavior-oriented design of autonomous agents," *From animals to animats*, vol. 3, pp. 445–452, 1994.

[13] R. Brooks, "Interaction and intelligent behavior," Ph.D. dissertation, Massachusetts Institute of Technology, 1994.

[14] O. Causse and L. Pampagnin, "Management of a multi-robot system in a public environment," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, vol. 2, 1995, pp. 246–252.

[15] F. Noreils, "Toward a robot architecture integrating cooperation between mobile robots: Application to indoor environment," *The International Journal of Robotics Research*, vol. 12, no. 1, pp. 79–98, 1993.

[16] E. Nowicki and C. Smutnicki, "An advanced tabu search algorithm for the job shop problem," *Journal of Scheduling*, vol. 8, no. 2, pp. 145–159, 2005.

[17] V. T'kindt and J. Billaut, *Multicriteria scheduling: theory, models and algorithms*. Springer Verlag, 2006.

[18] D. Trietsch and K. Baker, "Minimizing the number of tardy jobs with stochastically-ordered processing times," *Journal of Scheduling*, vol. 11, no. 1, pp. 71–73, 2008.

[19] Y. Martínez, A. Nowé, J. Suárez, and R. Bello, "A reinforcement learning approach for the flexible job shop scheduling problem," *Learning and Intelligent Optimization*, pp. 253–262, 2011.

[20] A. Page and T. Naughton, "Framework for task scheduling in heterogeneous distributed computing using genetic algorithms," *Artificial Intelligence Review*, vol. 24, no. 3, pp. 415–429, 2005.

[21] E. Opiyo, E. Ayienga, K. Getao, W. Okello-Odongo, B. Manderick, and A. Nowé, *Game theoretic multi-agent systems scheduler for parallel machines*. Fountain Publishers, Kampala., 2008.

[22] M. Wang, Z. Shi, and W. Jiao, "Dynamic interaction protocol load in multi-agent system collaboration," *Multi-Agent Systems for Society*, pp. 103–113, 2009.

[23] M. J. Mataric, "Interaction and intelligent behavior," Ph.D. dissertation, MIT EECS, 1994.

[24] B. Galitsky, J. de la Rosa, and B. Kovalerchuk, "Discovering common outcomes of agents' communicative actions in various domains," *Knowledge-Based Systems*, vol. 24, no. 2, pp. 210–229, 2011.

[25] A. Rawal, P. Rajagopalan, and R. Miikkulainen, "Constructing competitive and cooperative agent behavior using coevolution," in *IEEE Conf. on Computational Intelligence and Games*, 2010.

[26] M. Wang, H. Wang, D. Vogel, K. Kumar, and D. Chiu, "Agent-based negotiation and decision making for dynamic supply chain formation," *Engineering Applications of Artificial Intelligence*, vol. 22, no. 7, pp. 1046–1055, 2009.

[27] A. Chopra and M. Singh, "An architecture for multiagent systems an approach based on commitments," in *Int. Conf. on Autonomous Agents and Multiagent Systems, Workshop on Programming Multiagent Systems*, Budapest, Hungary, 2009.

[28] B. An, V. Lesser, and K. Sim, "Strategic agents for multi-resource negotiation," *Autonomous Agents and Multi-Agent Systems*, vol. 23, no. 1, pp. 114–153, 2011.

[29] M. Dias, R. Zlot, N. Kalra, and A. Stentz, "Market-based multirobot coordination: A survey and analysis," *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1257–1270, 2006.

[30] C. Clark, T. Bretl, and S. Rock, "Applying kinodynamic randomized motion planning with a dynamic priority system to multi-robot space systems," in *IEEE Aerospace Conf. Proceedings*, vol. 7, 2002, pp. 7–3621.

[31] E. Ferranti, N. Trigoni, and M. Levene, "Brick&Mortar: an on-line multi-agent exploration algorithm," in *Proc. IEEE Int. Conf. on Robotics and Automation*, 2007, pp. 761–767.

[32] A. Marjovi, J. G. Nunes, L. Marques, and A. T. de Almeida, "Multi-robot exploration and fire searching," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, St. Louis, MO, USA, 2009.

[33] Y. U. Cao, A. S. Fukunaga, and A. B. Kahng, "Cooperative mobile robotics: Antecedents and directions," *Autonomous Robots*, vol. 4, pp. 7–27, 1997.

[34] G. Dudek, M. R. M. Jenkin, E. Milios, and D. Wilkes, "A taxonomy for multi-agent robotics," *Autonomous Robots*, vol. 3, no. 4, pp. 375–397, 1996.

[35] D. Borrmann, J. Elseberg, K. Lingemann, A. Nüchter, and J. Hertzberg, "Globally consistent 3d mapping with scan matching," *Robotics and Autonomous Systems*, vol. 56, no. 2, pp. 130–142, 2008.

[36] J. Faigl and L. Přeučil, “Self-organizing map for the multi-goal path planning with polygonal goals,” *Artificial Neural Networks and Machine Learning*, pp. 85–92, 2011.

[37] H. Tamimi, H. Andreasson, A. Treptow, T. Duckett, and A. Zell, “Localization of mobile robots with omnidirectional vision using Particle Filter and iterative SIFT,” *Robotics and Autonomous Systems*, vol. 54, no. 9, pp. 758–765, 2006.

[38] S. Carpin and E. Pagello, “An experimental study of distributed robot coordination,” *Robotics and Autonomous Systems*, vol. 57, no. 2, pp. 129–133, 2009.

[39] F. Beutler, M. Huber, and U. Hanebeck, “Probabilistic instantaneous model-based signal processing applied to localization and tracking,” *Robotics and Autonomous Systems*, vol. 57, no. 3, pp. 249–258, 2009.

[40] D. Bhadauria, O. Tekdas, and V. Isler, “Robotic data mules for collecting data over sparse sensor fields,” *Journal of Field Robotics*, vol. 28, no. 3, pp. 388–404, 2011.

[41] G. Pfister, *In search of clusters*. Prentice-Hall Englewood Cliffs, NJ, 1998.

[42] T. Anderson and D. Culler, “A Case for Network of Workstations (NOW),” *IEEE Micro.*, vol. 15, no. 1, pp. 54–64, 1994.

[43] H. Casanova and J. Dongarra, “NetSolve: A network-enabled server for solving computational science problems,” *International Journal of High Performance Computing Applications*, vol. 11, no. 3, p. 212, 1997.

[44] K. Kurihara, S. Hoshino, K. Yamane, and Y. Nakamura, “Optical motion capture system with pan-tilt camera tracking and real time data processing,” in *Proc. IEEE Int. Conf. on Robotics and Automation*, 2002.

[45] F. Tagliareni, M. Nierlich, O. Steinmetz, T. Velten, J. Brufau, J. Lopez-Sanchez, M. Puig-Vidal, and J. Samitier, “Manipulating biological cells with a micro-robot cluster,” in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2005.

[46] E. Yoshida, S. Murata, A. Kamimura, K. Tomita, H. Kurokawa, and S. Kokaji, “A motion planning method for a self-reconfigurable modular robot,” in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2002.

[47] A. Şamiloğlu, V. Gazi, and A. Koku, “Effects of asynchronism and neighborhood size on clustering in self-propelled particle systems,” *Computer and Information Sciences*, pp. 665–676, 2006.

[48] R. Buyya, “High performance cluster computing: Architectures and systems (volume 1),” *Prentice Hall, Upper SaddleRiver, NJ, USA*, vol. 1, 1999.

[49] S. Georgiev, “Evaluation of cluster middleware in a heterogeneous computing environment,” Master’s thesis, Internationaler Universitätslehrgang, 2009.

[50] L. Dagum and R. Menon, “OpenMP: an industry standard API for shared-memory programming,” *IEEE Computational Science Engineering*, vol. 5, no. 1, pp. 46–55, 2002.

[51] A. Geist, *PVM: Parallel virtual machine: a users’ guide and tutorial for networked parallel computing*. the MIT Press, 1994.

[52] N. Desai, R. Bradshaw, A. Lusk, and E. Lusk, “MPI cluster system software,” *Recent Advances in Parallel Virtual Machine and Message Passing Interface, Springer*, pp. 277–286, 2004.

[53] J. Corbalan, A. Duran, and J. Labarta, “Dynamic load balancing of MPI+ OpenMP applications,” in *IEEE Int. Conf. on Parallel Processing*, 2004, pp. 195–202.

[54] Q. Snell, G. Judd, and M. Clement, “Load balancing in a heterogeneous supercomputing environment,” in *Int. Conf. on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, NV, 1998, pp. 951–957.

[55] M. Bhandarkar, L. Kale, E. de Sturler, and J. Hoeflinger, “Adaptive load balancing for MPI programs,” *Computational Science-ICCS*, pp. 108–117, 2001.

[56] G. Utrera, J. Corbalán, and J. Labarta, “Dynamic load balancing in MPI jobs,” in *High-Performance Computing*. Springer, 2008, pp. 117–129.

[57] K. Erciyes and R. Payli, “A cluster-based dynamic load balancing middleware protocol for grids,” *Advances in Grid Computing-EGC*, pp. 436–436, 2005.

[58] Y. Zhu, J. Guo, and Y. Wang, “Study on Dynamic Load Balancing Algorithm Based on MPICH,” in *IEEE WRI World Congress on Software Engineering*, vol. 1, 2009, pp. 103–107.

[59] I. Stoica, F. Sultan, and D. Keyes, “Modeling communication in cluster computing,” in *Proc. of the 7th SIAM Conf. on Parallel Processing for Scientific Computing*, San Francisco, CA, 1995, pp. 820–825.

[60] R. Martin, A. Vahdat, D. Culler, and T. Anderson, “Effects of communication latency, overhead, and bandwidth in a cluster architecture,” in *IEEE Int. Symp. on Computer Architecture*, 1997, pp. 85–97.

[61] J. Hromkovič, *Communication complexity and parallel computing*. Springer-Verlag New York Inc, 1997.

[62] M. J. Quinn, “Parallel Programming in C with MPI and OpenMP,” *McGraw-Hill Press*, 2003.

[63] R. Grabowski, L. Navarro-Serment, C. Paredis, and P. Khosla, “Heterogeneous teams of modular robots for mapping and exploration,” *Autonomous Robots*, vol. 8, no. 3, pp. 293–308, 2000.

[64] S. Thrun, “A probabilistic on-line mapping algorithm for teams of mobile robots,” *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 335–363, 2001.

[65] J. Ko, B. Stewart, D. Fox, K. Konolige, and B. Limketkai, “A practical, decision-theoretic approach to multi-robot mapping and exploration,” in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, vol. 4, 2003, pp. 3232–3238.

[66] G. Dedeoglu and G. Sukhatme, “Landmark-based matching algorithm for cooperative mapping by autonomous robots,” in *Int. Symp. on Distributed Autonomous Robotics Systems*, 2000.

[67] J. Fenwick, P. Newman, and J. Leonard, “Cooperative concurrent mapping and localization,” in *Proc. IEEE Int. Conf. on Robotics and Automation*, vol. 2, 2002, pp. 1810–1817.

[68] H. Wang, M. Jenkin, and P. Dymond, “Enhancing exploration in graph-like worlds,” in *IEEE Canadian Conf. on Computer and Robot Vision*, 2008, pp. 53–60.

[69] J. Jennings, C. Kirkwood-Watts, and C. Tanis, “Distributed map-making and navigation in dynamic environments,” in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, vol. 3, 1998, pp. 1695–1701.

[70] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes, “Topological exploration with multiple robots,” in *Int. Symp. on Robotics and Applications*, Anchorage, AK, USA, 1998.

[71] K. Konolige, D. Fox, B. Limketkai, J. Ko, and B. Stewart, “Map merging for distributed robot navigation,” in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, vol. 1, 2003, pp. 212–217.

[72] A. Marjovi, J. Nunes, L. Marques, and A. de Almeida, “Multi-robot fire searching in unknown environment,” in *Field and Service Robotics*. Springer Tracts in Advanced Robotics, 2010, vol. 62, pp. 341–351.

[73] C. Kruskal, L. Rudolph, and M. Snir, “A complexity theory of efficient parallel algorithms,” *Theoretical Computer Science*, vol. 71, no. 1, pp. 95–132, 1990.

[74] G. Amdahl, “Validity of the single processor approach to achieving large scale computing capabilities,” in *Spring joint computer Conf.* ACM, April 1967, pp. 483–485.

[75] F. ALECU, “Performance analysis of parallel algorithms,” *Journal of Applied Quantitative Methods*, p. 129, 2007.