DEPLOYMENT OF HETEROGENEOUS SWARM ROBOTIC AGENTS USING A TASK-ORIENTED UTILITY-BASED ALGORITHM

Tamer Abukhalil

Under the Supervision of Dr. Tarek M. Sobh

DISSERTATION

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIRMENTS

FOR THE DEGREE OF DOCTOR OF PHILOSOHPY IN COMPUTER SCIENCE

AND ENGINEERING

THE SCHOOL OF ENGINEERING

UNIVERSITY OF BRIDGEPORT

CONNECTICUT

JANUARY, 2015

DEPLOYMENT OF HETEROGENEOUS SWARM ROBOTIC AGENTS

USING A TASK-ORIENTED UTILITY-BASED ALGORITHM

Approvals

Committee Members

Name	Signature	Date
Dr. Tarek Sobh	T. Sobh	03/25/2015
Dr. Zheng (Jeremy) Li	Z. Jeremy Li	03/24/2015
Dr. Xingguo Xiong	Xingguo Xiong	03/06/2015
Dr. Miad Faezipour	Miad Faezipour	03,06,2015
Dr. Devdas Shetty	Devdas Shetty	03/16/2015

Ph.D. Program Coordinator

Dr. Khaled Elliethy	Khaled Elleithy	3/22/2015

Chairman, Computer Science and Engineering Department

Dr. Ausif Mahmood	Ausif Mahmood	3-6-2015

Dean, School of Engineering

Dr. Tarek Sobh	T. Sobh	3/25/2015

DEPLOYMENT OF HETEROGENEOUS SWARM ROBOTIC AGENTS USING A TASK-ORIENTED UTILITY-BASED ALGORITHM

DEPLOYMENT OF HETEROGENEOUS SWARM ROBOTIC AGENTS USING A TASK-ORIENTED UTILITY-BASED ALGORITHM

ABSTRACT

In a swarm robotic system, the desired collective behavior emerges from local decisions made by robots, themselves, according to their environment. Swarm robotics is an emerging area that has attracted many researchers over the last few years. It has been proven that a single robot with multiple capabilities cannot complete an intended job within the same time frame as that of multiple robotic agents. A swarm of robots, each one with its own capabilities, are more flexible, robust, and cost-effective than an individual robot.

As a result of a comprehensive investigation of the current state of swarm robotic research, this dissertation demonstrates how current swarm deployment systems lack the ability to coordinate heterogeneous robotic agents. Moreover, this dissertation's objective shall define the starting point of potential algorithms that lead to the development of a new software environment interface. This interface will assign a set of collaborative tasks

to the swarm system without being concerned about the underlying hardware of the heterogeneous robotic agents.

The ultimate goal of this research is to develop a task-oriented software application that facilitates the rapid deployment of multiple robotic agents. The task solutions are created at run-time, and executed by the agents in a centralized or decentralized fashion. Tasks are fractioned into smaller sub-tasks which are, then, assigned to the optimal number of robots using a novel Robot Utility Based Task Assignment (RUTA) algorithm. The system deploys these robots using it's application program interfaces (API's) and uploads programs that are integrated with a small routine code. The embedded routine allows robots to configure solutions when the decentralized approach is adopted. In addition, the proposed application also offers customization of robotic platforms by simply defining the available sensing and actuation devices. Another objective of the system is to improve code and component reusability to reduce efforts in deploying tasks to swarm robotic agents. Usage of the proposed framework prevents the need to redesign or rewrite programs should any changes take place in the robot's platform.

ACKNOWLEDGEMENTS

My thanks are wholly devoted to God who has helped me all the way to complete this work successfully. I owe a debt of gratitude to my family for their understanding, support, and encouragement.

I am honored that my work has been supervised by Prof. Tarek Sobh. He has taught me how good research work is done. I appreciate all his contributions of time, ideas, and funding to make my PhD experience productive and stimulating. The joy and enthusiasm he has for his research is contagious and motivated me, even during tough times throughout my PhD pursuit.

I would like to extend my special thanks to my friend and colleague, Dr. Sarosh Patel for his inspiring ideas. I am so grateful for his support and helpful comments. I am extremely thankful for the excellent example he has been as a researcher and a teacher. I wish him all the very best in his future.

I would like to convey my thanks to my friend and fellow PhD candidate Madhav Patil for the effort he has put in developing the physical robotic swarm agents. I would not have got the chance to complete my research without the actual hardware he has built.

NOMENCLATURE

R	Set of Robotic Agents
T	Task to be performed
n	Number of robots
r_i	Robot i
t_j	Subtask (j) Beginning Time
t'_j	Subtask (j) Ending Time
m	Number of Subtasks
v_{ij}	Subtask (j) performed by Robot (i)
$type$	Subtask type
t_{ej}	Subtask (j) Assigned Time
$rate_j$	Subtask (j) Consumption Rate
L	Workload
d_{ij}	Distance of Robot i to Subtask j
β_i	Robot Representation
id	Robot id
w_i	Robot (i) Wheel Slip
$Prem_i$	Robot (i) Power Remaining
t'_m	Total time to execute all Subtasks
u_1, u_2	Weight Coefficients
P_i	Solution Plan for robot (i)
pri	Priority

ACRONYMS

API	Application Program Interface
ANN	Artificial Neural Network
ACO	Ant Colony Optimization
CNP	Contract Net Protocol
DRRS	Dynamically Reconfigurable Robotic System
DoF	Degrees of freedom
EKF	Extended Kalman Filter
FPGA	Field Programmable Gate Array
GA	Genetic Algorithms
GP	Generic Programming
HDRC	Hormone Driven Robot Controller
HAL	Hardware Abstraction Layer
IR	Infra Red
JVM	Java Virtual Machine
MRTA	Multi Robot Task Allocation
OAP	Optimal Assignment Problem
PSA	Particle Swarm Algorithm
PWM	Pulse Width Modulation
PFSA	Probabilistic Finite State Automata
PIM	Platform Independent Model
RPC	Remote Procedure Call
RUTA	Robot Utility based Task Assignment

RFID	Radio Frequency Identification Tags
SOA	Service Oriented Architecture
SLAM	Simultaneous Localization And Mapping
SOAP	Simple Object Access Protocol
WS	Web Services

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE: INTRODUCTION

Decentralized modular robotics is an emerging area that has attracted many researchers over the last few years. The desired tasks may be too complex for one single robot, whereas they can be effectively done by multiple robots [1, 2]. Modular robotic systems have proven to be robust and flexible [3-7]. These properties are likely to become increasingly important in real-world robotics applications. When investigating the control environments that actually deploy such robotic systems to perform the intended tasks, our findings indicate a lack of software packages exist that provide control for various platforms of robots, individually, and allow concurrent control of heterogeneous robotic teams. Thus designed such control applications. Figure 1-1 shows the break-down of the system hierarchy.

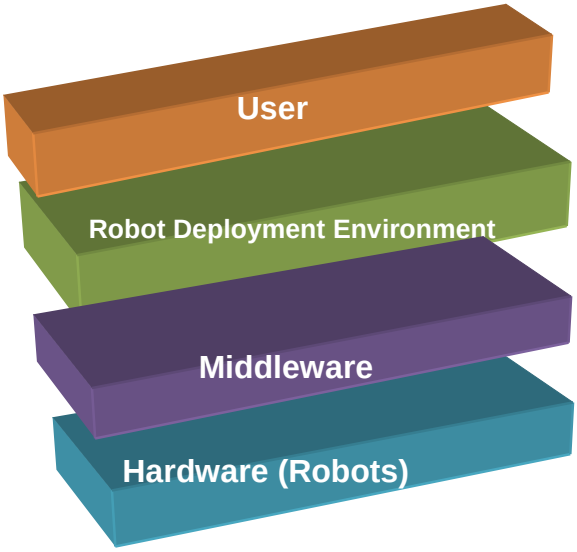


Figure 1.1: System Layers

Over the past decade, various research efforts have been performed that attempt to resolve coordination and decision making problems in swarm robotic systems. Such studies include simple models such as foraging [8, 9]. The multi-agent robotics system consisting of a number of identical robots proposed in [10] for a decentralized robot is yet another approach to swarms. In [11], Roderich and others proposed the concept of self-assembling capabilities of the self-reconfigurable S-bots (also known as "Swarm-bot") were developed by the Francesco Mondada et al. [12]. Swarm-bots can either act independently or self-assemble to form a swarm by using their grippers. In [13], Fukuda and Nakagawa proposed the concept of the DRRS (Dynamically Reconfigurable Robotic System) based on a cell structure for removable parts. The implementation was then called CEBOT, the first cellular robotic system. CEBOT is a heterogeneous system comprised of agents with different locomotion functions. One of the critical aspects of this type of system is the communication between the members of the swarm [14], which is usually carried out using radio-links. In [15], Dumbar and Esposito studied the problem of maintaining communications among the robots performing tasks.

Decentralization means that the algorithm does not require access to the full global state and all control computations are done locally. However, in order to command large groups of robots, it is also essential to include an element of centralization to allow humans to interact and task the team. Our work is based on the assumption that there is a lack of software packages which provide control for the different platforms of robots individually, and allow concurrent control of heterogeneous robotic teams.

1.1 Problem Scope

The scope of our research is making heterogeneous robots exhibit swarm behavior despite their different configurations. The first objective of our research is to develop a task-based software environment to deploy such robots and program them to operate in a swarm fashion. Our second research objective is to design an intelligent coordination component that generates optimal platform-independent algorithms to perform three essential tasks based on the parameters and the type of robots entered by the user. Since the trend in the swarm robotic research was mainly focused on developing rather large and homogenous systems, our work focuses on the development of smaller and less intelligent robots and having a large number of such systems to perform collaborative tasks.

1.2 Motivation

The number of such smaller agents is the key factor that has to be decided to answer the question, would a single robot with larger computational power complete an intended job using the same time and accuracy that of multiple robotic agents or not?

Our research hypothesis is based on two primary motivations. Our first motivation is to develop the necessary framework that will provide connectivity between heterogeneous agents, in addition to building central software that interacts with these agents. There are six software packages that are primarily designed to distribute programs and deploy the swarm of robotic agents. These packages provide simple communication, and allow for interfacing with the swarm of robots. Furthermore, these software packages

simulate the swarm systems and deploy tasks to the robots. Our system design is motivated by our interest in multi-robot control for the deployment of potentially large numbers of cooperating robots to perform tasks such as simultaneous navigation, object manipulation, and transportation. Such system design would be a great practical integration tool to provide rapid implementation of real-world experimentations without spending excessive time on writing software programs. Our Second motivation is to extend the software programs that are uploaded on each robot in order to allow them to integrate more sensors and/or actuators. These programs will allow auto-detection of the attached standardized components as they are added to current robot configuration. Having such programs will add a plug-and play feature to every robotic agent in the swarm system.

1.3 Research Contributions

Our software deployment environment (UBSwarm) is developed to simulate, deploy, and coordinate robots in real-time. By incorporating the improvements listed below, our research overcomes the limitations found in the existing multi-agent software deployment environments.

1. UBSwarm is designed to deploy heterogeneous robotic agents that have different hardware configurations and functionalities, unlike the previous swarm systems in which robotic agents were homogenous.
2. UBSwarm defines a set of operating rules and constructs programs that make the different robotic agents work in a swarm fashion even though the robots'

hardware configurations are different.

3. The proposed application offers customization of robotic platforms by defining the available sensing devices, actuation devices, and the required tasks. In addition, our application works to prevent rewriting programs for the different robotic configurations.
4. The service-oriented architecture used in our deployment system demonstrates how programs are constructed and how the coordination agent generates solution plans using the agent's Robot Utility Based Algorithm (RUTA).
5. UBSwarm is designed to deploy and coordinate the swarm robots by using either centralized or decentralized modes depending on the intended task.
6. By performing a premature simulation of the task, UBSwarm chooses the exact number and type of the mobile agents.

CHAPTER TWO: LITERATURE SURVEY OF SWARM SYSTEMS

In conducting our survey, we identified a criteria that is based on assumptions similar to the ones presented in [16]. We investigated the deployment systems that target algorithms designed to operate heterogeneous/homogenous robots performing various tasks. These assumptions can be summarized as follows:

1. The identified systems are composed of an undetermined number of embodied robots;
2. Heterogeneous robots have different capabilities;
3. These robots have a decentralized control;
4. Additional robots may be added to the system at any time;
5. Robots are multi-purpose, not task specific;
6. A coordination model should exist to guide the different robots.

In this literature review, we present a comprehensive study on the behavior of existing swarm systems dedicated to deploy different tasks/applications on collective and mobile reconfigurable robotic system. The modules used in these systems are fully autonomous mobile robots that, by establishing physical connections with each other, can organize into modular robots. We do not consider any particular hardware or

infrastructure of each swarm agent. Our work focuses on building control mechanisms that allow the system to operate several simple heterogeneous agents.

This literature survey is organized as follows: Section 2.1 provides a comprehensive survey of two primary swarm approaches, a "biologically inspired" robots, and "functionally inspired" robots. Section 2.2 presents a comparison between existing reconfigurable robots. In section 2.3, we discuss self-replicating robots. Section 2.4 analyses of the existing robotic software deployment systems.

2.1 Two Main Categories of Swarm Behaviors

Swarm behavior was first simulated on computers in 1986 using the simulation program Boids [17]. This program simulated simple agents (Boids) that were only allowed to move according to a set of basic rules set by programmers. These rules are in fact, algorithms known as the Particle Swarm Algorithms (PSA's). The model was originally designed to mimic the flocking behavior of birds, but it can also be applied to schooling fish and other swarming entities.

Different studies of complexity have been carried out over these types of systems [7, 14, 15, 18-24]. There have been many interpretations of the understanding and modeling of swarming behavior. Some researchers have classified these behaviors into two primary types namely biologically inspired and functionally built robots [25], while others have proposed two fundamentally different approaches that have been considered for analysis of swarm dynamics. These are spatial and non-spatial approaches [26]. In the first approach, "biologically inspired", designers try to create robots that internally

simulate, or mimic, the social intelligence found in living creatures. The second approach, “functionally inspired”, use task-specific designed robots generally engineered with constrained operational and performance capabilities which include sensors, grippers, and so on. Consequently, these artificial robots may only need to generate certain effects and experiments with the environment, rather than having to withstand deep scrutiny for “life-like” capabilities [5].

2.1.1 “Biologically Inspired” Robots

Multiple researchers have shown some interest in the foraging and other insect inspired coordination problem and have investigated these behaviors and summarized them into algorithms. Others were interested in exploiting swarm robots in the tasks of localization [18], surveillance, reconnaissance [19], and hazard detection [20, 21]. Pheromone-trail-based algorithms sometimes have the ability to dynamically improve their path [22] and can adapt to a changing terrain [27]. Ant-inspired foraging has been implemented in robots by various groups. One major difficulty can be exhibited in implementing the pheromone itself. Others have resolved problems of how robots should interact in the swarm. There have been many approaches dedicated to this:

1. *By means of physical markers*, where robots physically mark their paths in multiple ways, such as depositing of a chemical alcohol on the ground [22], drawing lines onto the floor using pen and paper [21], laying trails of heat [23], storing the pheromone values radio Frequency Identification Tags RFID [24], or emitting ultraviolet light onto a phosphorescent paint [28].

2. *Transmitting wireless signals when laying virtual landmarks in a localization space*. In the work of Vaughan et al., robots maintain an internal pheromone model with trails of waypoints as they move, and share it with other robots over a wireless network [27].

3. *Virtual pheromones* that consist of symbolic messages tied to the robots themselves rather than to fixed locations in the environment. In their experiment [19], the virtual pheromone is encoded as a single modulated message consisting of a type field, a hop-count field, and a data field. Messages are exchanged between robots through infrared transmitters and receivers. It is assumed that the robots receiving the pheromone can measure the intensity of the IR reception to estimate their distance from the transmitter.

4. *Foraging allocation ratio among robots*. In [29], Wenguo Liu et al, presented a simple adaptation mechanism to automatically adjust the ratio of foragers to resting robots (division of labor) in a swarm of foraging robots and hence maximize the net energy income to the swarm. Three adaptation rules are introduced based on local sensing and communications. Individual robots use internal cues (successful food retrieval), environmental cues (collisions with teammates while searching for food) and social cues (team-mate success in food retrieval) to dynamically vary the time spent foraging or resting.

5. *Dynamic programmed deployable beacons.* The method described in [30] provides local rules of motion for swarm members that adhere to a global principle for

both searching and converging on a stationary target in an unknown and complex environment via the use of immobile relay markers.

The survey does not span the entire field of intelligent swarm behavior robotics. Instead, it focuses on systems for which new algorithms for communication between robots have been demonstrated. Such algorithms can be found in the work of the following researchers:

1. Algorithm for Self-Organized Aggregation of Swarm Robotics using Timer: As a solution to self-organization among swarm agents, Xinan Yan, et al. [1] have proposed an aggregation algorithm based on some constraints for which neither central control nor information about locations of the agents are pre-given. The author's control strategy contains two states, Search and Wait for each individual robot as given in the model of probabilistic Finite State Automata (PFSA). Their algorithm assigns unique IDs to each robot. Knowing the total number of robots, randomly placed robots walk in the arena looking for other robots. Based on IR sensing and wireless connection capabilities installed on each robot, each can identify the others robot's ID. The group of encountered robots forms an aggregate, in which the robot with the larger ID defines the aggregate's characteristics and also insures that all robots in a particular aggregate must have the same timers. When the timer terminates, the robot tries to detach from its current aggregate. In the experiment, all the robots are identical. Each robot is mobile with limited ability of interaction including IR sensing for detecting objects and wireless communication for communicating with other robots.

2. Two foraging algorithms using only local communications: Nicholas R. Hoff et al. [31], have proposed two algorithms for searching the environment for an object of interest (food) and then returning this object to the base, keeping in mind that all robots do not have any prior information about the location of the food. Their algorithms are inspired by the foraging behavior of ants in which they mark paths leading from the nest to food by depositing a chemical pheromone on the ground. Ants use the distribution of the pheromone to decide where to move. In their first algorithm, two simple floating-point values are used such that some robots will decide to stop their normal search and become 'pheromone robots' at any given point. Those robots will act like locations of virtual pheromones. Other robots can read the pheromone level by receiving a transmission from the pheromone robot, and they can "lay" the virtual pheromone by transmitting to the pheromone robot. So, if there were a network of pheromone robots, the walker robots could use the distribution of virtual pheromone they were able to sense in order to decide how to move. If integer values are used instead of floating-point values at each virtual pheromone such that the nearest robot to the nest stores the digit 1 and the other robot that is close enough to communicate with the first robot, stores and transmits the digit 2. A walker robot can use these values to find a path to the nest by always moving to the lowest cardinality it detects.

2.1.2 “Functionally Inspired” Robots

Another line of swarm-based research can be found where robot agents are built to achieve specific tasks such as path finding using algorithms that are not necessarily based on imitating biological swarm organisms. In their previous work, Wang Bei, et al. [32] implemented what they call a robotic termite agent, which is able to simulate the wood-chip collecting behavior of termites. The authors have developed a software and hardware solution based on the simulation of collective building of a 2D termites’ colony. The termites (swarm of robot agents) gather wood-chips into piles following a set of predefined rules. Boe-Bot Robots are used. The Boe-Bot is built on an aluminum chassis that provides a sturdy platform for the servomotors and printed circuit board and comes with a pair of whiskers and gripper. Their tasks include moving on smooth surfaces, detecting new objects, dropping the woodchips and then picking up such objects as they are encountered. The robot agent turns for a 360 degrees angle until it detects an object. The robot then carries the object, holds it, and moves forward. The robot keeps holding the chip as it wanders in the environment until it detects another object (which is another woodchip). After releasing the object, the robot moves backward, turns at an angle of 45 degrees, and the same procedure is repeated.

Obtaining decentralized control that provides interesting collective behaviors is a central problem [16, 33-41]. Several algorithms have been developed to run on swarms of robots. The complexity varies between these algorithms. Some provided basic functionality, such as dispersion, while others exhibited complex interactions between the team of robots such as bidding on tasks according to some rules. Table 2-1 summarizes

the most recent swarm robot systems with their corresponding algorithms. These are systems introduced in literature that only involve multiple agent teams with decentralized control.

Table 2.1: Multi-robot Coordination Approaches

Approach			Remarks
Approach 1 **Knowledge-based coordination**			
Symprion/ replicator	What determines the behavior of either single or group of agents is HDRC (Hormone Driven Robot Controller) controller that contains a configuration for the robot itself, and a software controller called Genome. The Genome contains a set of rules that control each agent's behavior and generates different actions according to the different environmental conditions. Agents keep learning about their environment using internal, external and virtual sensors. Agents also are supported with on-board computational power using approaches like Generic Programming (GP) and Genetic Algorithms (GA).	Kernbach et al., 2008 [41]	The most primary advantage of this approach is the huge number of units used in the experiment. Moreover, These modules are able to reassemble different shapes that could get the whole structure moving to desired locations.
iRobot	Authors suggest spreading pheromones in an ad-hoc way over the wireless network constituted by the robots. The primary communication component is	J.McLurkin and J.Smith, 2004 [33]	Their solution mainly focuses on path planning and routing protocols of messages transmitted

	an infrared inter-robot communication. Swarm software is written as behaviors that run concurrently. Each behavior returns a variable that contains actuator commands. Their goal is to spread robots throughout an enclosed space quickly and uniformly, that were identified by direct dispersion performed by two algorithms. The first one works by moving each robot away from the vector sum of particular positions from their closest neighbors. In the second one, robots move towards areas they have yet to explore. Once the robots know their positions the frontier robots issue a message. The trees created by these messages guide the swarm toward the frontier robots.		between agents at their different positions. However, the cost of individual robots and the number of robots required to provide sufficient coverage to the environment are high. This particular system suffers from the fact that when the ad-hoc network of robots gets partitioned, pheromone trails automatically break down causing the robots to stop moving.
Quadrotors	Authors attempt to design small light weight flying vehicles designed to operate in close ranges. The team of quadrotors is organized into groups. Vehicles within the group are tightly coordinated. Centralized control and planning is possible. The inter-group coordination is not centralized. Each group is controlled by a dedicated software node, running in an independent thread.	A.Kushleyev, et al., 2012 [42]	Quadrotors rely on an external localization system for position estimation and therefore cannot be truly decentralized

Approach 2: **Auction-based coordination**			
Layered architectures coordination	Authors propose auctions in which a bidding process takes place among the agents to determine who will be 'foreman' and will be in-charge for a given task and to secure teammate participation in subtasks. Tight coordination is implemented using an inexpensive reactive approach. Each robot consists of a planning layer that decides how to achieve high-level goals, an executive layer that synchronizes agents, sequences tasks and monitors task execution, and a behavioral layer that interfaces with the robot's sensors and effecters. Robots execute plans by dynamically constructing task trees.	R. Simmons, S. Singh, D. Hershberger, J. Ramos, and T. Smith, 2000 [34]	The three robots used in this experiment are coordinated by a manipulation manager which means this is a centralized system.
ASyMTRe-D	The authors' approach is based on schemas such as perceptual and motor schemas. Inputs/outputs of each schema create what it is called semantic information that is used to generate coalitions. Tasks are assigned to the robot with the highest bid. Bids are calculated according to the costs of performing different tasks. A set of tasks is allocated to coalitions. Coalition values are calculated based on the task requirement and robot capabilities. Execution of tasks is monitored and the process of allocation	Tang and Parker, 2007 [43]	The advantage of this approach is that it enables robots to adopt new task solutions using different combinations of sensors and effecters for different coalition compositions. However, that solution is mainly related to computational performance where tasks are static. Authors do not mention

	repeats itself until each individual task is completed. During run-time their novel protocol ASyMTRe-D takes place. This protocol manipulates calculated coalition values to assist in completing tasks.		the dynamical tasks and ways of task reassignment. Additionally, they do not discuss fault tolerance, flexibility, robustness, and how the system reacts to any robot failure.
RoboCup 2002 (Sony legged league)	Authors used wireless communication between robots in a 4-player soccer team. Each robot broadcasts a message to its teammates. This message contains the current position of the robot and some other information about the ball in that position. All of the robots use the same set of functions to calculate real valued bids for each task. Once each robot calculates the bids for itself and each of its teammates, it compares them. If it has the highest bid for the role being assigned, it assumes that role. If it was not the winner, it assumes that the winning robot will take up the role and performs calculations for the next role in the list.	D. Vail and M. Veloso,. 2003 [35]	Communications between robots is critical for successful coordination between robots. Local information about the field will not be enough. This approach does not coordinate a large scale of robots.
Another application of soccer robots.	Authors use dynamic role assignment as in Robocup basing on information gathered from best behavior. Two	E. Pagello et al. 2006 [36]	

	intermediate levels have been provided to allow robot individuals to communicate. The lower level implements stigmergy (indirectly stimulating the performance of the upcoming action to provide coordination between agents) whereas, the higher one deals with the dynamic role exchange. Authors use schema-based methodology. They discuss all perceptual schemas with the required sensing, also feeding the C-implemented motor schemas which demand immediate sensor data Robots are equipped with unidirectional cameras.		
M+ scheme for multi robot allocation and corporation	Each robot considers all currently available tasks at each iteration. For each task, each robot uses a planner to compute its utility and announces the resulting value to the other robots. Robots negotiate which one will be in charge of performing the task. For these tasks, robots create their own individual plans and estimate their costs for executing these tasks. The robots then compare their costs to offers announced by other robots.	S. Botelho and R. Alami, 1999 [37]	Relying on Negotiation Protocols, may complicate the design of the coordinating system. Furthermore, such negotiation scenario can drastically increase communication requirements/overhead.

MURDOCH a general task allocation system	The coordination system works using an auction protocol that allocates tasks via a sequence of first-price one round auctions. Every auction is issued by agents in five steps: task announcement, metric evaluation, bid submission, close of auction, progress monitoring/contract renewal. For each task auction, each available robot broadcasts its bid. Because of the asymmetric nature of MURDOCH's auctions, the running time varies between the bidders and the auctioneer. Authors two main testing domains were a long-term scenario consisting of many loosely coupled single-robot tasks, and a cooperative box-pushing task requiring tight coordination among the robots.	Brian P. Gerkey and Maja Mataric, 2002 [16]	M+ and MURDOCH systems assume that each robot has a single task. Each task may be performed by a single robot. This assumption proves to be oversimplified as many task domains require simultaneous work from multiple robots.
Market-economy Approach	Authors define three strategies for exploring unvisited regions. In the first strategy namely random goal point selection the goal points are chosen at random and discarded if the area surrounding the goal point has already been visited. In the second one, the goal point is centered in the closest unexplored spot as a candidate exploration point. In the last strategy,	R. Zlot, A. Stentz, M. B. Dias, and S. Thayer, 2003 [38]	Authors consider regions of potential target locations for each robot and distribute tasks using bid auctions. According to some experiments performed in [44], this approach

	the region is divided into its four children if the fraction of unknown space within the region is above a fixed threshold. Robots are initially placed into known positions. While running, each robot will try to sell each of its tasks to all robots with which it is currently able to communicate via an auction. If two robots lie in the same region, the robot with the highest bid wins that region's task.		could be useful if the number of robots is small compared to the number of frontier cells. However, in the case of multiple robots this approach can be disadvantageous since a robot discovering a new frontier during exploration will often be the best suited to go on it. This can lead to an unbalanced assignment of tasks and increased overall exploration time.

2.2 Reconfigurable Robots

Reconfigurable robots automatically rearrange and change their shape accordingly to adapt themselves to different environments of application. Reconfigurable robots exhibit some features that make it possible for the robots to adapt to different tasks. For example shape shifting robots could form a worm-like shape to move through narrow spaces, and reassemble into spider-like legged robot to cross uneven terrain. Another important feature of modular robots is their potential for self repair. As the modules making a unit up are usually identical, it is possible to eliminate the damaged module and substitute it using another one, if available. Modular robots are usually composed of

multiple building blocks of a relatively small repertoire, with uniform docking interfaces that allow transfer of mechanical forces and moments, electrical power, and communication throughout the robot.

According to M. Yim et al. [39], modular self-reconfigurable robotic systems can be generally classified into three architectural groups based on the geometric arrangement of their units. The first group consists of lattice architectures where robot units are arranged and connected in some regular, three-dimensional pattern, such as a simple cubic or hexagonal grid. The second group consists of chain/tree architectures where units are connected together in a string or tree topology. Finally, the third group consists of mobile architectures where units use the environment to maneuver around and can either hook up to form complex chains or lattices or form a number of smaller robots that execute coordinated movements. A respectable number of self-reconfigurable robot systems have been proposed in the last decade. Table 2-2 shows comparisons between the most recent ones.

Table 2.2: Comparisons between existing reconfigurable robot systems

Robot	**Author**	**Learned Pros and Cons**	**Software**	**Units**	**Communication**
SuperBot (2006)	Shen et al.[40]	Decentralized control. Reliable Mechanical design. Limitations: Infrared sensors limit the search range and require line-of-sight between SuperBots. SuperBot architecture lacks extra actuators, grippers, and sensors for gathering information about the working environment.	Low-level programs written in C and Real-time java-based operating system	3D Modules	Infra-red and a wireless capability limited to some functions
Molecubes (2005)	Zykov et al.[45]	Molecubes are low cost, small lattice based swarm robot with 3 DOF. Limitations: Unable to provide heavy object transport. Limited sensors. Lacks actuator mechanism.	2-D simulation	Cubes with 120 swiveling	None
YaMor (2006)	R. Moeckel et al. [46]	Each module comprises an FPGA for more computational power. Limitations: Uses onboard low-capacity batteries that limit the usefulness of modules. Limited sensors limit ability to sense surroundings. Only two controllable arms	Java-based GUI connected to robots via wireless connections	3D Chain of modules	Bluetooth
Swarm-bot (2006)	Groß et al. [11]	Robot swarms consisting of 2 to 40 S-bots have been successfully demonstrated. S-Bots are fully autonomous	Neural Networks	S-bots with grippers	No communications occur between

		mobile robots capable of self-navigation, perception of the environment and objects. Capable of communicating other S-Bots and transporting of heavy objects over very rough terrain. Limitations: Initial cost is high. Images and sound are the only way of communicating with other S-Bots. Large number of sensors and actuators consumes power, reducing functionality and operating time.			individual robots (S-bots) however each s-bot connects wirelessly to the PC.
Catom (2005)	Goldstein et al. [47]	Largest actuated modules (many electromagnets on modules) Limitations: Limited sensors that have limited ability to sense surroundings.	NA	3D Massive volume of agents ($m^{3)}$	This papers only presents a principle so no actual implementation
M-TRAN (2002)	Murata et al. [48]	Very small actuated modules, highly-robust, miniature, and reliable. Quick self-reconfiguration and versatile robotic motion. Limitations: Connection mechanism works on an internally balanced magnetic field that is not strong enough to hold the other modules. Single M-	OpenGL Library, M-TRAN simulator	3D Double-Cubes	Serial bilateral communications to the PC.

		TRAN module does not have enough DOFs for switching from one posture to another form. Lack of sensors leads to mapping and control problems. Power consumption is more as it uses servo motor and electromechanical force for connectivity.			
ATRON (2004)	E. H. Østergaard et al. [49]	Each module is equipped with its own power supply, sensors and actuators, allowing each module to connect and communicate with a neighbor module. Able to sense the state of its connectivity and relative motion. Limitations: Since each module includes two-axis accelerometers only, a module cannot tell if it is turned upside down or not. When two modules are connected, it's very difficult for them to move themselves, which requires cooperation from its neighbor. They are not mechanically stable and due to this mechanical instability, their electronic performance is poor.	On-board system	Lattice type units	Infra-red diodes
PolyBot (2002)	Yim et al. [50]	First system to demonstrate the ability of self-reconfiguration with most	NA	Lattice	Infra-red Interface

		active modules in a connected system. Each module fits within the 5cm cube. They are versatile in nature. Each module contains a Motorola PowerPC 555 processor with 1MByte of external RAM, and DC brushless motor with built in hall effect sensors. Limitations: Insufficient sensory unit for mapping of environment. Cannot work in unknown environment with rough surface or when obstacle avoidance is not possible.			
Replicator /Symbrion (2008)	7th framework program project, European Communities [41]	Multiple processors for different tasks. Limitations: Limited to a specific task. Lack actuators and connection mechanisms to physically attach to other modules.	On-board system	Lattice/ Chain	N/A

2.3 Self-replicating Robots

Designing fully autonomous replicating systems did not come true until the early 2000's. An attempt to design semi-autonomous self-replicating robots that demonstrated the LEGO Mindstorm kits as a prototype capable of replication under human supervision

was introduced in [51]. An autonomous self-replicating robot consisting of four low-complexity modules was presented in [52]. The authors proposed a system composed of a parent robot, four unassembled modules, and an environment in which the self-replication takes place. They defined two operations namely expansion and separation in which the parent robot grows itself by attaching the resource modules onto itself until it doubles its physical size, and then splits in the middle thereby returning the parent to its original state and producing one more robot. The parent robot is made of four cube-like modules connected to each other with electromagnets (EMs) installed in female and male couplers.

In [53], similar work has been done, also using unassembled components placed at certain locations on a track. The authors presented a robot that can assemble exact functional self-replicas from seven more basic parts/subsystems. The robot follows lines on the floor using light sensors and a simple control circuit without any onboard memory.

2.4 Swarm Control Software Environments

Trifa V. et al., [54] have proposed a methodology that supports standardized interfaces and communication protocols which connects robots produced by different manufacturers. The authors have used the so-called Service Oriented Architecture (SOA) in which different software components exchange data over HTTP and then create Web Services (WS). The authors proposed a system that consists of four parts namely, the physical layer which contains the actual e-puck robots, the gateway layer which acts like a connection between the physical devices and the system, the logical layer containing a

server that runs on J2EE, and the interface layer which provides services to the end users. In their system, any physical device or program capable of running HTTP such as PDAs, Tablet PC, and mobile phones can interact with the interface regardless of the operating system on the device. (No further explanation about control modules or how the interface looks like was given in the article). The e-puck robot –the standard one- has eight infrared proximity and light sensors, a triangular microphone array, a speaker, a three-axis accelerometer, and a Bluetooth interface for programming. The e-puck platform can be upgraded with custom pluggable modules such as the short-range radio communication turret which provides a subset of the 802.15.4 and ZigBee protocols and is fully interoperable with the MicaZ nodes used in the physical gateway layer. However, using SOA has some performance limitations as it requires a sophisticated messaging infrastructure that would restrict the capabilities of software running on robots.

Kulis et al., [55] have proposed a software framework for controlling multiple robot agents by creating a Distributed Control Framework (DCF). DCF is an agent-based software architecture that is entirely written in Java and can be deployed on any computing architecture that supports the Java Virtual Machine. DCF is specifically designed to control interacting heterogeneous agents. DCF uses a high-level platform-independent programming language for hybrid control called MDLE. The DCF architecture consists of two distinct agents: a Robot Agent and a Remote Control Agent (RCA). The RCA lies within the human interface shown in Figure 2-1. Robot Agents process data from onboard hardware and from other agents, and react to perceived stimuli

by selecting an appropriate behavior which is a sequence of control laws with embedded state transition logic according to a mission plan. Using the RCA, the end user can select tasks for either a robot agent or a group of agents using simple drag and drop operators. When agents are in place, a popup menu appears prompting the user to select a task. Relevant tasks for a team mission are defined in an XML configuration file which is loaded by the RCA at startup. The XML file also specifies which tasks can be performed by each agent. The authors also added a simulating feature to their RCA agent which provides a flexible numerical solving integrating system that solves differential equations for simulating a robot's kinematics/dynamics. Another feature of this system, it provides automatic updating of sensors and actuators to be distributed across multiple computing resources. The DCF currently provides drivers for a variety of robots (e.g., iRobot Creates, Pioneers, Amigobots, FireAnt, LAGR), and a wide range of sensors (e.g., digital encoders, Sonars, stereo cameras, GPS receivers, and inertial navigation systems)

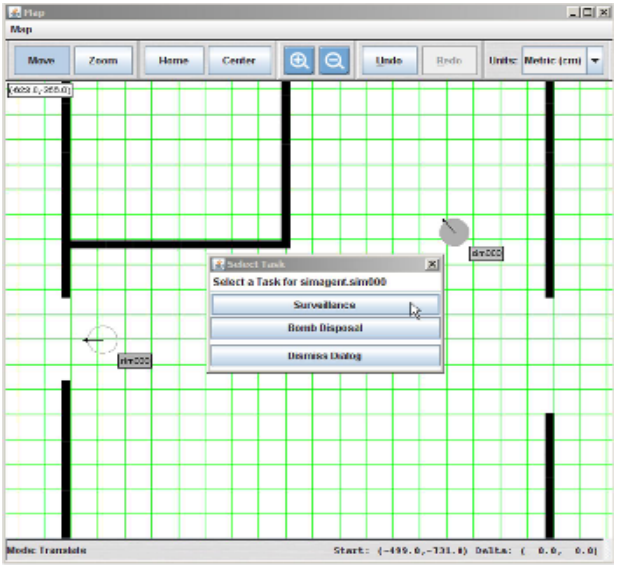


Figure 2.1: The DCF human interface (© 2008 IEEE)[1]

[1]© [2008] IEEE, Permission granted by Mr. Babak Sadjadi [55].

Multiple efforts have been conducted as part of enhancing the DCF system. Other versions of the DCF called JAUS and TENA are being developed and tested [56].

Gregory P. Ball G. et al. [8], have proposed application software built in JAVA to operate heterogeneous multi-agent robots for the sake of educational purposes named MAJIC. The system provides basic components for user interaction that enables the user to add/remove robots change the robotic swarm configuration, load java scripts into robots and so on as shown in figure 2-2. The system establishes communications with built-in robot servers via a wireless connection that uses the client/server relationship. The authors described their architecture as components, consisting of one higher level component that is the GUI manager, two application logic components that consist of a logical layer to parse input into valid commands, and a robot server, which receives commands from the logical layer and communicates these commands to the appropriate robot. Local components communicate using direct procedure calls.

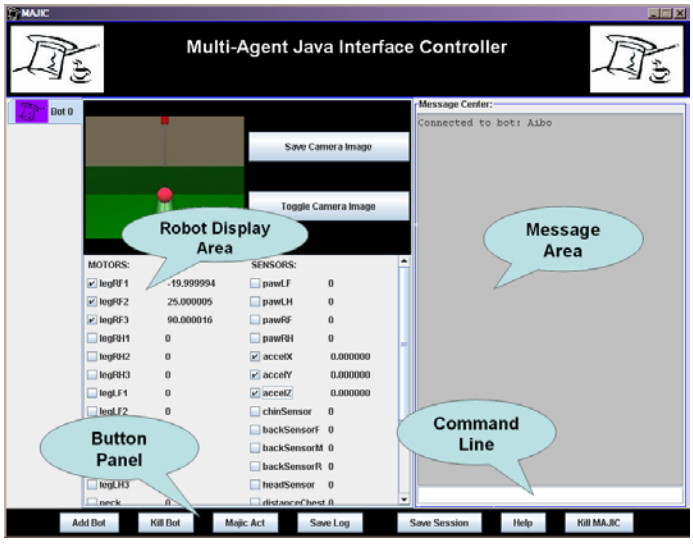


Figure 2.2: MAJIC Control Platform (© 2008 IEEE)[2]

[2]© [2008] IEEE, Permission granted by Dr. Craig Martell [8].

In order to operate robots, the user needs to write Java-embedded programs that use either the MAJIC library or Java libraries. Once a robot is connected to MAJIC, the user can immediately communicate with it from the command line. However, repeating this process for a team of heterogeneous robots can be impractical. The MAJIC system does not allow the user to specify the types of sensors a robot is equipped with or the type of motion model the robot's move command will utilize. This would allow the user to develop more intricate behaviors with greater precision.

In [57], Patricio Nebot et al., were more interested in developing cooperative tasks among teams of robots. Their proposed architecture allowed teams of robots to accomplish tasks determined by end users. A Java-based multi-agent development system was chosen to develop their proposed platform. The authors used *Acromovi* architecture which is a distributed architecture that works as a middleware of another global architecture for programming robots. It has been implemented by means of the MadKit (Multi-Agent Development Kit) multi-agent systems framework. The graphical interface is built around pure Java Swing components, thus resulting in a cross platform application, capable of running in any operating system running the Java virtual machine.

Tao Zhang et al. [58], proposed a software platform comprised of a central distributed architecture that runs in a network environment. Their system is composed of four parts namely, user interface, controlling center, robot agent, and operating ambient making up the platform top-down. The user interface is deployed on a terminal anywhere as long as it can connect to the server where the control center is deployed. The control center provides Application Program Interfaces APIs for users. The user interfaces

basically communicate with the control center via a network, using TCP/UDP protocol. Authors' platform was mainly developed using Java.

In robotic control environments, a graphical application software such as MobileEyes [59] and the C++ based software URBI [60] are available as open source systems. URBI provides GUI packages that aim to make compatible code to different robots, and simplify the process of writing programs and behaviors for these robots. URBI works by incorporating sensor data to initiate commands to the robot. URBI packages, however, provides no abstractions therefore they do not allow separating the controlling system from the rest of the system. For example, a control system might be intimately tied to a particular type of robot and laser scanner. Moreover the URBI's uniform programming language is limited to few kinds of microcontrollers available on the market. The Player/Stage proposed by Gerkey et. al. [61] also produces tools for simulating the behavior of robots without an actual access to the robots hardware and environment. Its two main products are the Player robot server, a networked interface to a collection of hardware device drivers, and Stage, a graphical, two-dimensional device simulator. The player/Stage is basically designed to support research in multi-robot systems through the use of socket-based communication. The player/Stage is open source software that is available to be downloaded online on UNIX-like platforms. However, running this software requires a variety of prerequisite libraries and each library requires another set of libraries. It has never been easy to understand how the system communicates with the actual robots. Player/Stage mainly supported robotic platforms such as RWI/iRobot, Segway, Acroname, Botrics, and K-Team robots.

Another script-based robot programming is Pyro[62]. Pyro, which stands for Python Robotics, is a robotics programming environment written in the python programming language. Programming robot behaviors in Pyro is accomplished by programming high-level general-purpose programs. Pyro provides abstractions for low-level robot specific features much like the abstractions provided in high-level languages. The abstractions provided by Pyro allow robot control programs written for small robots to be used to control much larger robots without any modifications to the controller. This represents advancement over previous robot programming methodologies in which robot programs were written for specific motor controllers, sensors, communications protocols, and other low-level features.

Ayssam Elkady et. al. [63] have developed a framework that utilizes and configures modular robotic systems with different task descriptions. Their main focus was designing a middleware that is customized to work with different robotic platforms through a plug-and-play feature which allows auto detection and auto-reconfiguration of the attached standardized components installed on each robot according to the current system configurations. Therefore, the authors' solution is mainly dealing with the abstraction layers residing between the operating system rather than software applications. A similar system hierarchy is used in Mobile-R [4] where the system is capable of interacting with multiple robots using Mobile-C library [64], an IEEE foundation for physical agents standard compliant mobile agent systems. Mobile-R provides deployment of a network of robots with off-line and on-line dynamic task allocation. The control strategy structure and all sub-components are dynamically

modified at run-time. Mobile-R provides some packages to enhance system capabilities like Artificial Neural Networks (ANNs), Genetic Algorithms (GAs), vision processing, and distributed computing. The system was validated through a real world experiment involving a K-Team Khepera III mobile robot and two virtual Pioneer2DX robots simulated using the Player/Stage system.

CHAPTER THREE: RESEARCH PLAN AND SYSTEM ARCHITECTURE

We are developing a software environment to utilize the heterogeneous robots that have different modular design, configuration of sensory modules, and actuators. The system will be implemented as a GUI interface to reduce efforts in deploying swarm robotic agents. The proposed application offers customization for robotic platforms by simply defining the available sensing devices, actuation devices, and describing the required tasks. The main purpose for designing this framework is to reduce the time and complexity of the development of robotic software and maintenance costs, and to improve code and component reusability. Usage of the proposed framework prevents the need to redesign or rewrite algorithms or applications when there is a change in the robot's hardware, operating system, or the introduction of new sensory/actuation units.

UBSwarm environment is a collection of high end APIs used for distributing algorithms to heterogeneous robotic agents. One of the key features of UBSwarm is configuring special programs which act as middleware that gain control over the agent's parameters and devices. The middleware consequently allows auto-detection of the attached standardized components according to current system configurations. These components can be dynamically made available or unavailable. Dynamic detection

provides the facility to modify the robot during its execution and can be used to apply patches and updates, to implement adaptive systems. This real time reconfiguration of devices attached to different robots and driver software makes it easier and more efficient for end users to add and use new sensors and software applications. In addition, the high-end interface should be written in a flexible way to get better usage of the hardware resource. Also they should be easy to install/uninstall. The general overview of the UBSwarm deployment platform and the overall system overview are shown in Figure 3-1 (a) and (b) respectively.

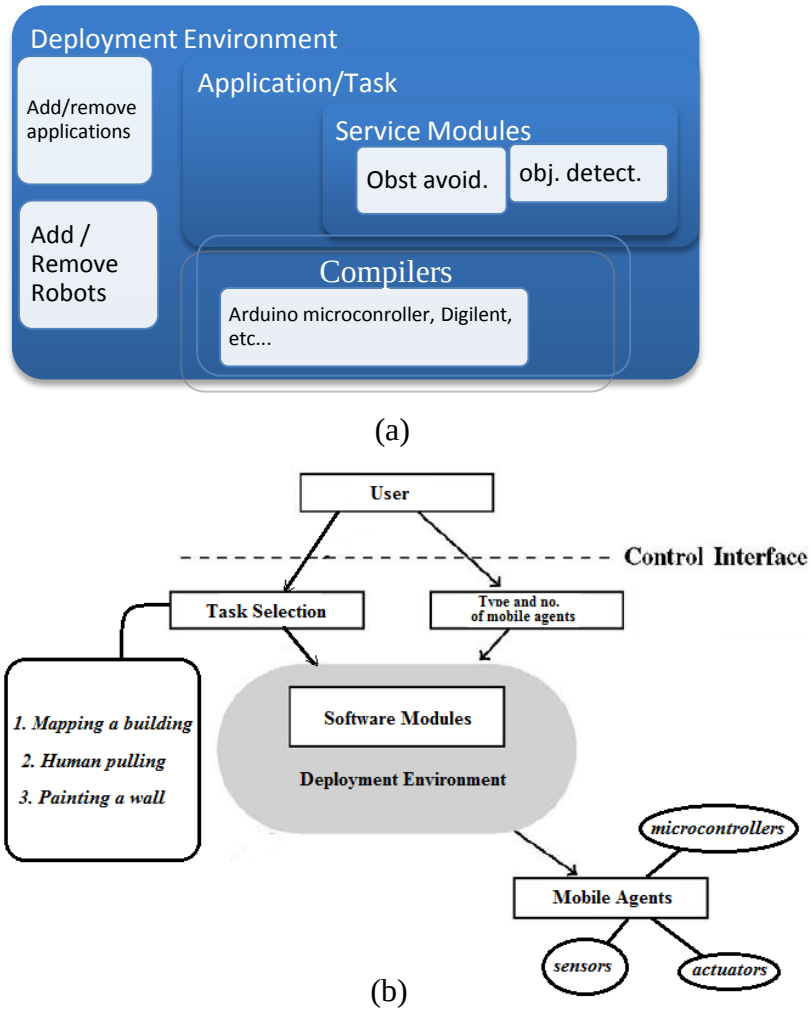


Figure 3.1: (a) Deployment software overview (b) System overview

Another key feature of the UBSwarm interface is to move the communication implementation from the user's domain to the application domain. Instead of learning proprietary protocols for individual robots, the user can utilize the UBSwarm scripting language to pass common commands to any robot managed by the application. UBSwarm adds a layer of abstraction to such tasks, allowing users the ability to intuitively obtain desired responses without extensive knowledge of robot-specific operating system and protocol. When users make changes to the hardware devices that are plugged onto the robotic agent, UBSwarm will provide the appropriate software package for these sensory devices and actuators. This flexibility makes it easy for the end users to add and use the new devices and consequently task applications. In addition, the software code can be written in the most common programming languages such as python, C#, or any programming language that is specific to a particular robot framework. These Software components are easy to install/upload in the console screen. At start up, UBSwarm uploads a code that is responsible for scanning for hardware changes onboard because almost all microcontrollers include a hardware feature to interrupt the current software routine and run a scanning routine when a particular pin changes states. By relying on the hardware to notice a change we can keep track of hardware components. Each one of these hardware component is operated using a particular algorithm that is created at the time of deployment. UBSwarm runs on a computer and uploads programs and monitors the robots through the USB (serial port), Radio Frequency (RF), WiFi, or Bluetooth. In our experiment we used our own robot agents that incorporate Arduino and Digilent Max32 microcontrollers.

UBSwarm provides a direct two-step configuration that helps the operator to select between several available robot microcontrollers, actuators, and sensors and then assign the group of robots a particular task from the set of predetermined tasks. To test and evaluate the swarm system or to change the configuration of the whole system, the user should be able to change each robot's features. That is, the user will have the option to add/remove hardware features of any selected robot. The user can also decide which robots to be assigned for the task. In the main menu, the user is given a list of tasks to be assigned to the swarm system. At the time of startup the system will expect the user to do either of the following two:

1- Configure the system by picking the available agents, their onboard components (sensors, motors, etc.) and the services needed to accomplish each task
2- Run the system using saved configurations and only allow add/remove agents.

UBSwarm is an interactive Java-based application designed for extensibility and platform independence. The system establishes communications with embedded robot modules. As shown in figure 3-2, the system is divided into two main subsystems, a robot deployment system and a robot control and translation system. The robot control system includes a robot control agent in which the user should provide all the parameters required for all sensors incorporated on robots. The user should also describe actuation methods used. The robot deployment system encapsulates a variety of high-level applications module which contains the tasks the platforms perform such as navigation, area scanning, and obstacle avoidance. A hardware abstraction layer is used to hide the

heterogeneity of lower hardware devices and provide an interface to communicate with robot platforms.

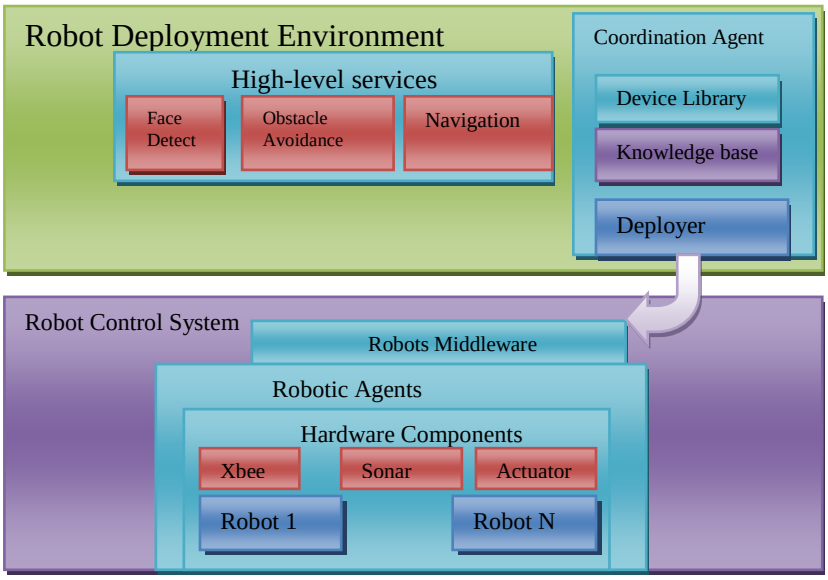


Figure 3.2: System architecture

3.1 Robot Deployment System

The deployment system takes responsibility of running actions according to the definition parameters and the integrations of the heterogeneous robots. Each application is implemented as a software module to perform a number of specific tasks used for sensing, decision-making, and autonomous action. Actions are platform independent robot algorithms; for example, it can be an obstacle avoidance algorithm or a data processing algorithm using Kalmans filter, etc. These actions can communicate together using message channels. The deployment system framework is shown in figure 3-2. The

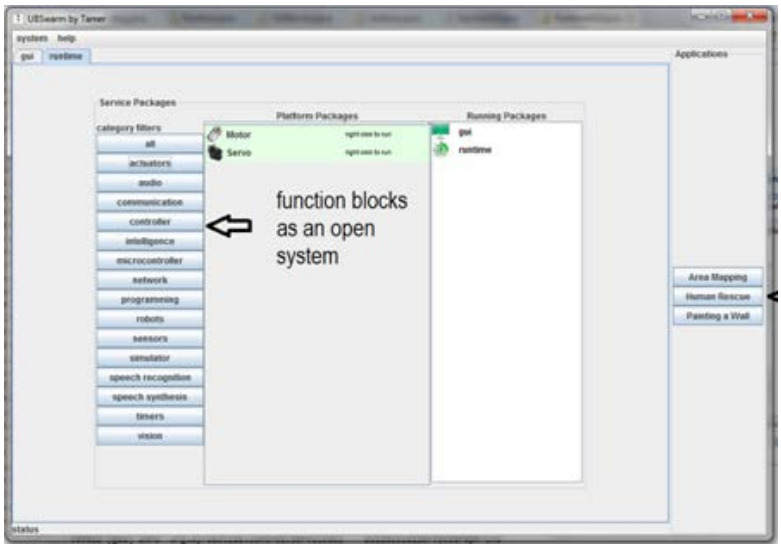
deployment system contains the developer interface, the coordination manager, the runtime interpreter, and the knowledge base.

3.1.1 User Interface

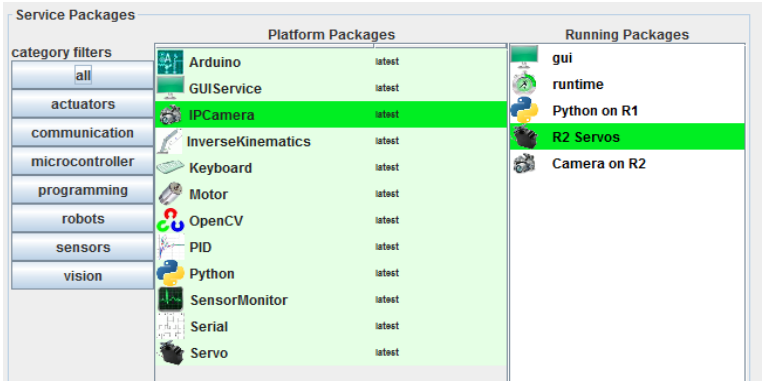
The system developer interface provides the human operator command and control windows. The user can interact with the computer through interaction tools which provides a list of application tasks and the available robotic agents. In the next windows, the user will be prompted to input the required system parameters for all sensors incorporated on robots such as the PIN location by which each sensor/actuator is connected to. As we mentioned earlier UBSwarm connects to the robots using either of USB cable, RF, WiFi, or Bluetooth. The user has to provide the IP address of the particular robot when WiFi is used. When connecting the robot to the USB, UBSwarm will detect the COM port automatically. After defining all required parameters, the user will have the chance to write programs and upload them on each robot. The interface provides a number of tasks that can be assigned to the group of robots such as SLAM, and human rescue (pulling an object). The interface also provides open system design that allows entering various new functions, tasks, robots and sensors. Each task is defined as functional modules. Obstacle avoidance, navigation, and SLAM are examples of such functional modules. Each functional module encapsulates services such as Opencv, Hough transformation, etc. Each service is regarded as a component of the system and is described in an XML configuration file to remove platform dependency. The user interface also allows the users to update, remove, or add robots in the swarm group. After clicking on a particular task, the user will be prompted to pick a number of

robots displayed in a list of the available robot types. The right set of button in figure 3-3 (a) is the group of available tasks available. The left set of buttons are designated for an open system extendibility which allows entering new tasks by simply altering the code embedded in each of the shown categories.

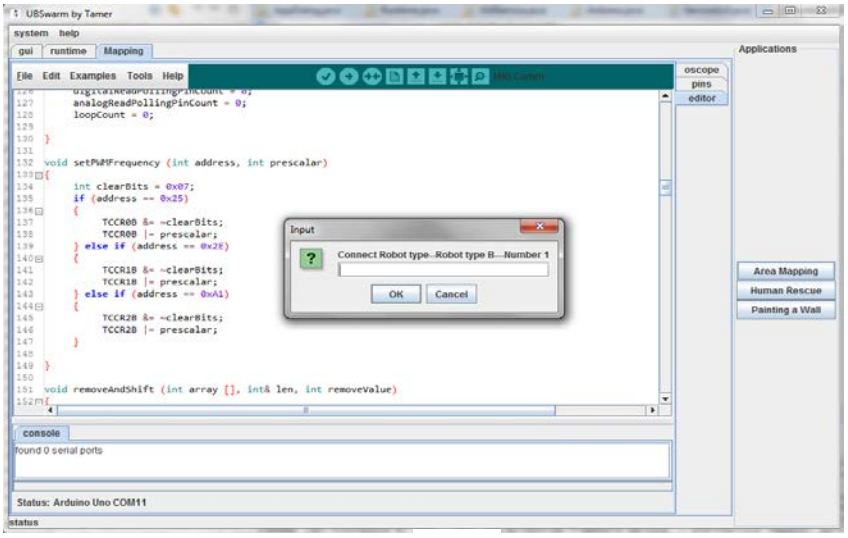
The user will be then be asked to enter each agent's initial pin locations (once for each type of robot) associated with various hardware components such as ultrasonic sensors, scan servo motors, and the n pin locations for the n-DoF arm if any is attached on the robot. A value of -1 will be assigned to pin locations of components that does not exist on the particular robot. The programs which will be uploaded on each robot type will differ according the different pin locations associated with each type that were set by the user. The system will ask the user to connect each robot to allow uploading the program as shown in fig. 3-3 (c). The next four subsystems show how the deployment system works to manage the heterogeneity of the hardware and the software associated with each robotic agent. Figure 3-3 (b) shows the coordination manager running in background as a running package (runtime) the figure also shows the service package that runs the object detection algorithm (camera on R2).



(a)



(b)



(c)

Figure 3.3: (a) List of available tasks (b) runtime coordinator and other running packages (c) Additional robot is added to the system

3.1.2 Coordination Manager

The heterogeneity of the robots and the operating platforms imposes dependencies such as data format, location of machine addresses, and availability of the components. Just like the functional modules contained in the coordination manager framework, the relevant tasks are stored in the knowledge base. Relevant tasks for a team mission are defined the XML configuration file which is loaded at startup. The XML file also specifies which tasks can be performed by each agent. The coordination agent processes the available state data and activates high-level behaviors using rules defined in a schema approach in order to select the appropriate robots and actions based on the provided tasks. The coordination agent framework comprised of five components: the Communication Protocol Module (CPM), task module, coordinator, task composer, and the deployer. The framework of the coordination manager is shown in figure 3-4.

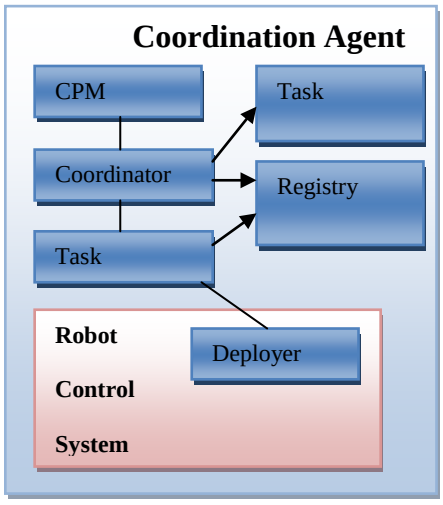


Figure 3-4: Coordination manager framework

A. The Communication Protocol Module (CPM)

This module stores communication access to all available communication devices and the necessary protocols used by the different hardware devices to communicate.

B. Task Module:

The task agent contains the necessary algorithms, data, and core functions to a given task for example obstacle avoidance.

C. Coordinator:

The coordinator utilizes the information given by the task module in order to select the appropriate robots and actions based on the provided sub-tasks.

D. Task Composer

Once the coordinator completes its task, the allocated tasks are broken down into required actions from actuator movements to communications.

E. Deployer

The deployer is the component responsible for sending the composed programs to the Robot Control System.

The algorithm used in the coordination manager is based on artificial intelligence approaches based on task allocation. A break-down of the algorithm used in the coordination manager is shown in figure 3-5.

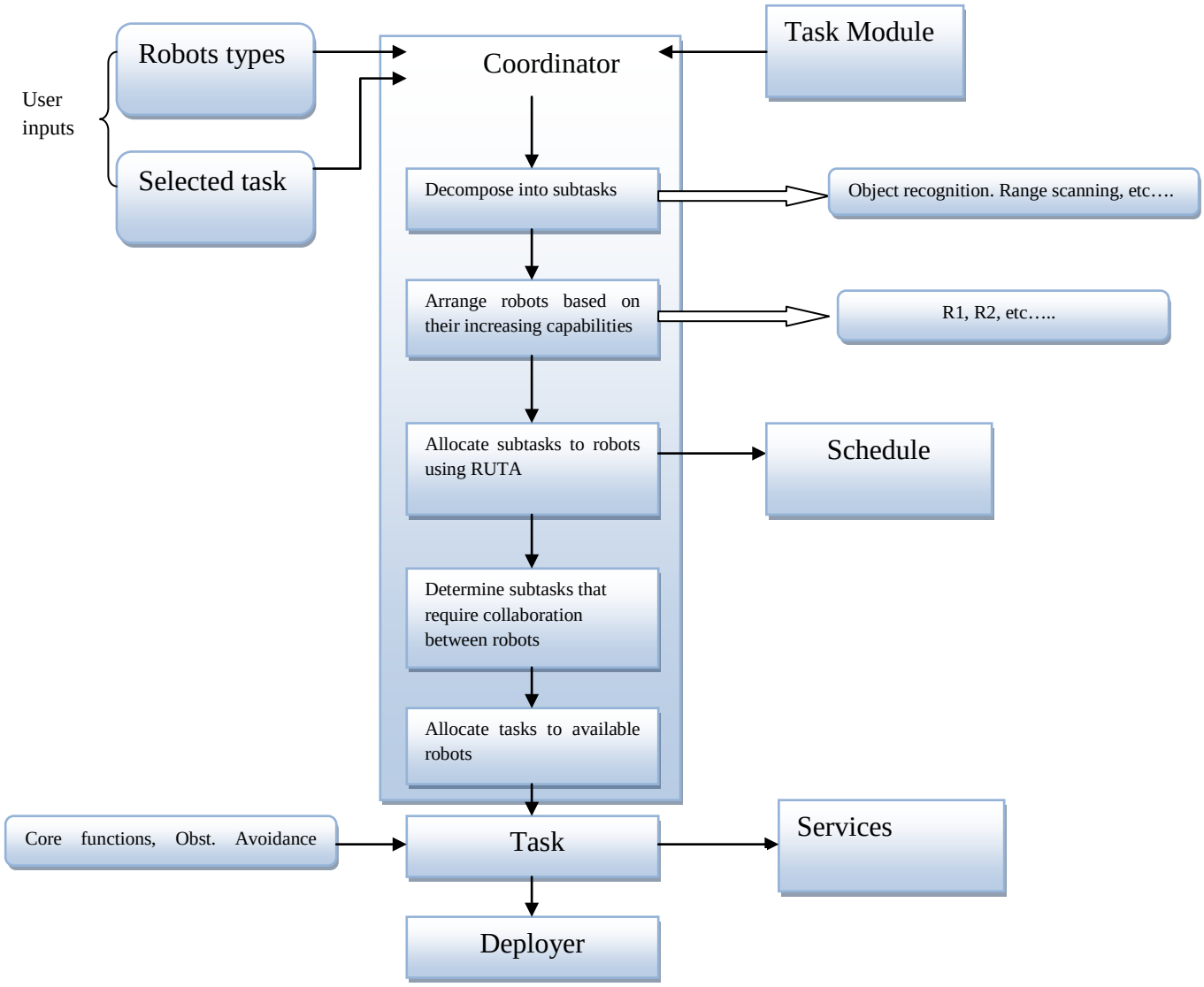


Figure 3-5: Coordination manager algorithm overview

3.1.2.1 Utility-based Solution for Optimal Task Assignment

To show how the system coordination manager generates solutions for an optimal number of robotic agents, we assume that $R = \{r_1, r_2, \ldots\ldots, r_n\}$ is a collection of n robots, where each robot r_i is represented by its available environmental sensors (ES), motor devices (MD), and communication devices (CD). For example, table 3-1 shows the

configuration of robots in the experiments. Table 3-2 shows the different consumption rates for the components integrated on the robots.

Table 3-1: Five robots and their capabilities

Robot	**Available sensor (s) /capabilities**	**Wheels slip percentage**
R1	VGA camera, URM Ping, V32 ultrasonic, 2-Dof arm, wheel Serial motors.	3%
R2	V32 Ultrasonic, 2-dof arm, two Serial motors	1%
R3	Sonar sensor, 1- Dof arm, two Serial motors	20%
R4	Serial motors, two sonar sensors, 1-Dof arm	5%
R5	VGA camera, Serial motors, two sonar sensors, 1-Dof arm	30%

Table 3-2: Sensing and actuation components consumption rates

Sensing/actuation Component	**Consumption rate**
VGA Camera	20 mA
URM Ping	20 mA
V32 Ultrasonic	4 mA
2-Dof (2 servos)	2x(120 mA)
Serial motors for wheels	2x(160 mA)

Our approach to multi-robot task allocation problem (MRTA) is based on the following assumptions:

- T is task to be accomplished, which is a set of *m* subtasks that are basically composed of motor, sensor and communication devices that need to be activated in certain ways in order to accomplish this task. Its denoted as

$T_i = \{v_{i1}, v_{i2}, v_{i3}, \ldots\ldots, v_{im}\}$ where v_{ij} is the subtask *j* performed by robot r_i and $1 \leq i \leq n, 1 \leq j \leq m$

- A subset v_{ij} of T_i, can be allocated to robots concurrently if they do not have ordering constraints.
- To accomplish the task T_i on robot r_i, a collection of *n* plans (solutions), denoted $P_i = \{P_1, P_2, \ldots, P_n\}$, needs to be generated based on the task requirements and the robot capabilities.

We define a cost function for each robot, specifying the cost of the robot performing a given task, and then estimate the cost of a plan performing the given task. We consider two types of cost:

- A robot-specific cost determines the robot's particular cost (e.g., in terms of energy consumption or computational requirements) of using particular capabilities on the robot r_i to accomplish a task v_{ij} (such as a camera or a sonar sensor). We denote robot $r_i's$ cost by *robot_cost*(r_i, v_{ij}).
- The cost of a plan P_i performing a task T_i is the sum of individual cost of n robots performing sub-tasks m that are in the plan P_i, which is denoted as: $Cost(Pi, Ti) = \sum_{i=0}^{n} \sum_{j=1}^{m} \text{cost}\left(r_i, v_{ij}\right)$

The problem we address here is the optimal assignment problem (OAP) which uses the Utility concept found in game theory [65]. Our solution is called Robot Utility-

based Task Assignment (RUTA) and it can be summarized as the following: given (T, R), find a solution P_i to each task T_i such that $Cost(Pi, Ti)$ is minimized.

We assume that sub-tasks $t_{j's}$ allocated to robot r_i must be ordered into a schedule $\sigma_i = (v_{i1}, t_1, t'_1), \ldots\ldots, (v_{ij}, t_j, t'_j)$ for $1 \leq j \leq m$ where v_{ij} is the subtask performed from time t_j to t'_j. Each sub-task assigned to a robot is denoted by a triple; $\alpha_j = < type, t_{ej}, rate_j >$ representing the v_{ij} task type whether its sensing or actuation type, time assigned to the task until it is accomplished (so $t_{sj} = t'_j - t_j$), and the consumption rate (in mA) for this particular subtask respectively. Depending on the robot $r_i's$ location, the time spent on each task must equal $r_i's$ assigned share of the workload. We also assume that the distance in meters between robot r_i and the location of the subtask v_{ij} is d_{ij}. Taking these values into account, each robot can be represented as $\beta_i = < id, w_i, Prem_i >$, representing the robot's id, percentage of wheel slip, and power remaining to perform the sub-task respectively. The mathematical quality of a robot r_i performing a subtask v_{ij} is calculated by dividing the robot r_i battery remaining power by the product of multiplying the sensor and/or actuator consumption rate with the percentage of time in which its operating. This is determined by the following equations

$$\varphi_{manip\ ij} = 0.7 \times \left[\left(\frac{t_{sj}}{t'_m} \right) \left[\frac{Prem_i}{rate_{act\ j}} \right] \right] \tag{3.1}$$

$$\varphi_{nav\ ij} = 0.7 \times \left[\left[\frac{prem_i}{rate_{servo\ j}} \right] \times \frac{1}{w_i} \right] \tag{3.2}$$

$$\varphi_{sens\,ij} = 0.9 \times \left[\left(\frac{t_{sj}}{t'_m} \right) \left[\frac{Prem_i}{rate_{sens\,j}} \right] \right] \tag{3.3}$$

$$\varphi_{given\,ij} = \varphi_{manip\,ij} + \varphi_{nav\,ij} + \varphi_{sens\,ij} \tag{3.4}$$

Where t'_m is the total time predetermined for the robot r_i to complete all of its subtasks in seconds, w_i is the pre-assumed percentage of robot r_i wheel slip, and $\varphi_{manip\,ij}$, $\varphi_{nav\,ij}$ and $\varphi_{sens\,ij}$ are the qualities to perform manipulating, navigation, and sensing subtasks respectively. Depending on the subtask type, the value of any of these quality functions is null if they are not taking place in the subtask. $\varphi_{gevin\,ij}$ is the total quality of subtask v_{ij} being performed by robot r_i. When obstacle avoidance task is being performed, the quality function $\varphi_{given\,ij}$ has higher values than the other qualities because it includes navigation as well as sensing subtasks. The priorities of subtasks must be considered and are calculated according to the schedule of tasks σ_i that is set to robot r_i. The priority of robot r_i performing a subtask v_{ij} is defined by equation (3.5) varying from 0 to 1.

$$pri_{ij} = \frac{1}{2} \times min\left[(u_1 \times (t - t_j), 1 \right] \tag{3.5}$$

Where t is the current time elapsed since the beginning of the task, t_j is the time when the task is announced as declared in the schedule σ_i. The parameter u_1 adjusts how the priority should increase with the value of $(t - t_j)$.The assignment of a subtask v_{ij} to the specific robot (that is capable of accomplishing it) is determined by the Utility function of a robot r_i performing a task v_{ij} as in the following equation:

$$utility_{ij} = \max(0,\ u_2 \times (d_{ij}{}^{-1/2} \times \varphi_{given\ ij} \times pri_{ij}) \quad (3.6)$$

Where $utility_{ij}$ is the nonnegative utility of robot r_i for sub-task v_{ij} , 1 <= *i* < =*n*, 1 <= *j* <= *m*, u_2 is the weighted coefficient to adjust the effect of the variables inside the equation. We assume that each robot r_i is capable of executing at most one task at any given time. We also assume that multiple agents can also share a single sub-task in which they divide the workload.

Initially the task is introduced to the system which performs the following set of algorithms.

Initialization Algorithm 3.1: Input: (T, R,M,N)

1. *Schedule sub-tasks v_j , such that ordering constraints are satisfied.*
2. *if (N=1) then Stop*
3. *Else*
4. *Sort the robots according to decreasing computational and sensory capabilities*
5. *Initially the $utility_{ij}$ for all robots and subtasks is equal to 0*
6. *Calculate utilities of each of the N robots*
7. *Based on the task requirement T, pick "at least" two robots with highest utility values.*
8. *For each sub-task v_j*
9. *For each robot r_i of the two selected robots*
10. *Assign subtask v_j to r_i based on the task requirements*
11. *→Add (r_i, v_j) to plan Pi*
12. *→Update parameters in v_j*

As the task is being executed the following two algorithms take place. The optimal number of robots is decided by running the following algorithm which is equal to the final value of i.

Centralized Algorithm 3.2: Input: (T, R,M,N)

1. *For each unexecuted sub-task* v_j *in the schedule*
2. *For each robot* r_i *in the new robot ordering*
3. *{ Calculate Utility function* $utility_{ij}$ *for robot* r_i
4. *If current utility of* r_i *for sub-task* v_j *is greater than its previous utility then assign subtask* v_j *to* r_i *based on the task requirements*
5. *→Add* (r_i, v_j) *to plan Pi*
6. *→Update parameters in* v_j

7. *Stop when the task is completed or after K number of trials*
8. *Go to step 10 if a faulty robots is discovered*
9. *}*
10. *If task is not complete, Pick a robot with the highest utility value from the list of remaining robots*
11. *→Add to robots ordering*
12. *Go to step 1*

In the distributed approach, decentralized coordinated programs are uploaded on the swarm of robots at start up. The programs allow the set of robots to reason, reassign, and execute subtasks later during their mission should a failure or a change in the swarm team is introduced. During runtime, each robot simply calculates its own utility when tasks are taking place as shown in *Algorithm 3*. Information about robot status (such as

any error readings from sensors) are shared between robots. If the task is interrupted or a failure is introduced to the swarm team, robots are able to reconfigure new task solutions to cope with changes in team composition and task requirements.

Decentralized Algorithm 3.3: Input (R,N)

1. *Utility is calculated on each robot*
2. *Two robots with highest utility values will begin their pre-programmed plans*
3. *While task is not complete*
4. *{*
5. *Each robot's utility value is shared with the other robots. When a robot is introduced to the system or If a sensor fails on one robot r_i by which it prevents it from completing task v_j, it sends a request (bid) to the other robots in the team.*
6. *Robot waits for reply (t_{out}) to hear respond from the most fit one (based on the winner highest utility value).*
7. *Task v_j is taken over by the winning robot.*
8. *}*
9. *Stop is task is complete else call the next robot in the ordering R*

In the decentralized approach, the coordination among robots is achieved through a distributed negotiation process based on sharing of information. The task allocation is achieved using a variant of the well known Contract Net Protocol (CNP) [66] with a slight alteration that information are shared between robots only when a task is interrupted or a failure is introduced to the swarm system. The solution is evaluated based upon each robot's local information, and the final decision is determined by mutual selection. The negotiation process is triggered at each failure to generate initial solution

strategies, and is called to re-plan solutions to accommodate changes in the robot group. It is important to note, however, that the distributed approach trades off solution quality for communication overhead.

3.1.3 Runtime Interpreter

As deployment programs are being constructed, the runtime interpreter calls new platform independent software packages specific for the execution of the sensory and actuation components associated with each agent. When new service is added to the system, the dynamic interpreter manages flow of information between these services by monitoring the creation and removal of all services and the associated static registries. The dynamic interpreter maintains state information regarding possible & running local services. The host and registry maps are used in routing communication to the appropriate tasks. The dynamic interpreter will be the first service created which in turn will wrap the real JVM runtime objects. When new services are added to the system, messages will be initiated by the runtime interpreter. The message consists of two basic parts: the header (which describes the data being transmitted, its origin, its data type, and so on) and the body (data). There are four types of messages, the *Command message*, used to invoke a service in another application; the *Document message*, used to pass a set of data to another application; the *Event message*, used to notify another application of a change in this application and the *Request-Reply message*, used when an application should send back a reply. The messages are classified into three categories: simple message (small messages with low delay requirements), realtime message (small message with a certain deadline), and message stream (message sequence with a certain rate). The

priority setting of a message can be adjusted an urgent message that should be delivered first. System developers can extend the system's functionality by adding new service modules to the list of available modules that can be found under the "runtime" tab in the main menu. Figure 3-6 shows the operation of the runtime interpreter when services are added to the system.

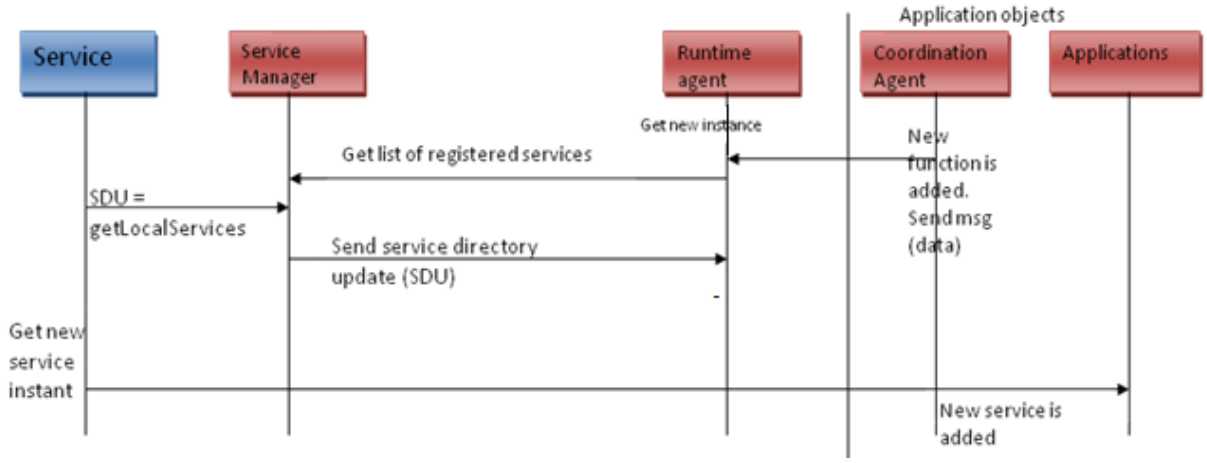


Figure 3-6: Adding services in runtime

Once the coordination agent completes its job, the dynamic agent breaks down allocated tasks into required actions from actuator movements to communications. Then, the dynamic interpreter monitors the flow of data, manages the flow of messages through the system, makes sure that all applications and components are available, tracks quality of service (e.g. response times) of an external service, and reports error conditions. The dynamic interpreter does its job by utilizing a component requirement matrix for each robot. The component requirement matrix is used to combine the necessary components from the knowledge base to the mobile agents which are then passed to the robot control and translation agent. As described in [63] each component has an XML configuration

entry to customize its behavior. Each component is designed to be dynamically reconfigurable by the dynamic interpreter during robot operation.

3.1.4 Registry

The registry contains all of the necessary information for each robot to give the coordinator the ability to address each task. This includes a listing of all possible actions, service modules, and behavioral components implementations for each robot. The registry stores service types, dependencies, categories and other relevant information regarding service creation. It also includes the agents' required communication protocols, and their drivers. Physical and logical addresses associated with each component are also stored in the knowledge base.

3.2 Robotic Control System

From programming prospective, the robot agent is a class. This class specifies the methods that must be provided by implementing such a class. The class interface architecture enables a loose coupling between the control algorithms and the underlying hardware; alternative hardware sensors supporting the required sensing functionalities may be interchanged freely (tested in the experiment). Unlike some robot agents that contains a regular PC as part of their systems; our swarm system is composed of robotic agents that incorporate onboard microcontrollers. UBSwarm supports most of the Arduino and Digilent PIC microcontrollers. Each robot has TX/RX pins which uses the microcontrollers' serial communication and turns it into IO-slave. Each robot agent incorporates two software programs to perform its job:

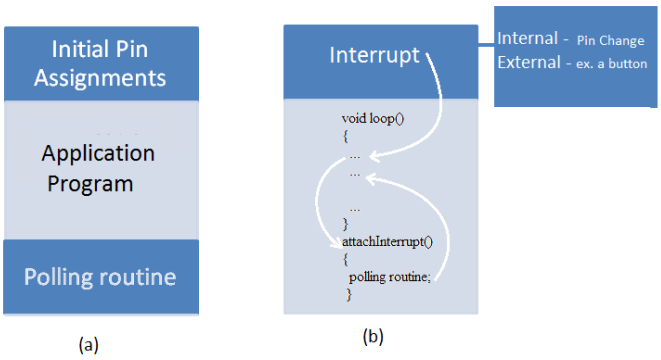


Figure 3-7: (a) controlling program, (b) Interrupt execution

3.2.1 Application Program

The program which is uploaded on each robot agent consists of the task-related controlling code, the initial pin assignments, and the polling routine as shown in figure 3-7 (a). This program contains function blocks to operate all the current hardware components which are currently connected and all possible functions associated with each new component that might be attached to the robot. The controlling program has some conditional statements to decide which function to call. The decision of which blocks of code to run depends on the updated pin assignments after the execution of the polling routine and the task intended from the robot. The polling routine is executed only if an internal interrupt has been activated.

3.2.2 Polling Routine

The polling routine is basically the hardware tracker/scanner of the robotic agent. It is a piece of code that resides within the application program; its job includes receiving raw data from onboard sensors. When an external interrupt is activated, the processor takes immediate notice, saves its execution state, runs the polling routine, and then

returns back to whatever it was doing before. Fig 3-5 (b) shows the sequence of actions when internal or external interrupt is triggered. The type of interrupt used is an external button connected to an interrupt pin and the ground (GND). When this pin change its signal edge (from rising to falling or vice versa), the polling routine scans all the other signal pins for newly attached components. After gathering such data, the polling routine sends messages that include the state data about the hardware components attached to each I/O pins. This data also include the type of the sensor. In order for the polling routine to understand which kind of sensor has been connected, we divided the set of pins into two categories:

1- Digital PWM (Pulse Width Modulation) pins can only be connected to Ultrasonic sensors or servo motors

2- Analog pins can only be connected to Infra-red or sonar sensors

The polling algorithm can be summarized as follows:

Algorithm 3.4: Polling routine

1. *Initially some signal pins are connected to components*
2. *Main program begin*
   {
3. *Attach the Interrupt pin to the interrupt function*
4. *If (interrupt is activated) then goto polling routine*
   } Main program end
5. *Polling routine begin{*
6. *For each unassigned pin set its internal pull-up resistor to high*
7. *For each unassigned pin wait 1sec for change in signal*
8. *If signal change occur {*
9. *Add type of sensor and its pin number to vector array*
   }
10. *Update pin assignments } end polling routine*

The robot control middleware also incorporate the following module which provides essential input to the polling and controlling programs.

3.2.3 Device library

The device module contains information to be uploaded to the XML file about the hardware components which are classified according to the functionalities they provide. For example, a GPS receiver can function either as a position device or as a range device.

3.2.4 Compiler

The Compiler gets a single input from the CPM module of the coordination agent; this input is the type of the microcontroller board connected. Based on the board type, the compiler will have the information it needs to know about the microcontroller and the I/O ports. For example, the Arduino microcontroller boards have PIN arrangement as follows:

1- Serial: 0 (RX) and 1 (TX): These pins are used to receive (RX) and transmit (TX) transistor-transistor logic (TTL) serial data. For example on the Arduino Diecimila, these pins are connected to the corresponding pins of the FTDI USB-to-TTL Serial chip.

2- External Interrupts (pins 2 and 3): These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.

3- PWM Pins: 4 upto 24 Provide 8-bit PWM output.

4- Analog Pins: pins 25 and so on analog input pins support 10-bit analog-to-digital conversion (ADC)

Relevant tasks for a team mission are defined in the device library configuration file which is loaded by the UBSwarm at startup. The device library file also specifies which tasks can be performed by each agent and if applicable, the physical hardware sensors and devices to be used.

3.2.5 Hardware Abstraction Layer

The *Hardware Abstraction Layer* (*HAL*), is the platform dependent part of UBSwarm. It is used to hide the heterogeneity of lower hardware devices and provide a component interface for the upper layers call. *HAL* removes hardware and operating system dependencies between the robot and the application in order to assure portability of the architecture and application programs. It provides access to the sensor data or actuation commands abstracted from the underlying physical connection of the resource. The abstraction layer (HAL) as shown in Figure 3-2, contains wrappers to hardware dependent control libraries which act as a low-level middleware to hide the heterogeneity of the underlying microcontrollers.

CHAPTER FOUR: IMPLEMENTATION AND TEST PLAN

Three different application tasks have been implemented to test the software development software namely: simultaneous localization and mapping (SLAM), human rescue, and wall painting. The mobile robots are built at the Robotics, Intelligent Sensing & Control (RISC) Laboratory at the University of Bridgeport. The system is composed of five heterogeneous robots in the sense that they have different sensory and actuation components. The prototypes of the robots are shown in figure 4-1. The robots are built using Arduino UNO, Arduino Due, and Digilent PIC boards. These boards are designed to make the process of attaching hardware components easier.

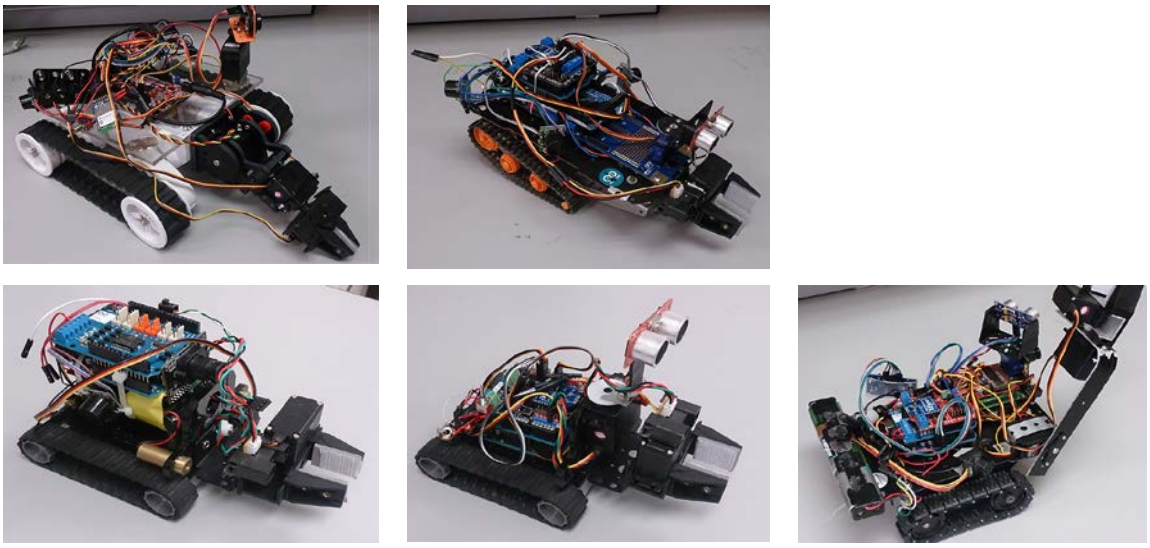


Figure 4-1: Heterogeneous robots showing different components

The hardware consists of a simple open hardware design and a rigid frame to support and secure the different types microcontroller boards and on-board input/output components. As for the power source, five packs of 7.2V 2500mAh Ni-MH batteries ensure sufficient energy autonomy to the robots. For distance sensing, URM V3.2 and PING ultrasonic sensor were used. However, as experimental results depict, the sensing capabilities of the platforms can be easily upgraded with other sensors, *e.g.,* laser range finders. Additionally, the platforms are also equipped with an Xbee Shield from Maxstream, consisting on a ZigBee communication module with an antenna attached on top of the Arduino Uno board as an expansion module. This Xbee Series 2 module is powered at 2mW having a range between 40m to 120m, for indoor and outdoor operation, respectively.

UBSwarm runs on a windows operating system. The deployed robots have simple behaviors and the overall high intelligence of the group is created by the simple acts and moderate local intelligence of each individual robot. Each customized program contains parameters that will be initially assigned to default values when starting UBSwarm interface. The three application tasks used in evaluating UBSwarm environment can be summarized as follows:

4.1 Simultaneous Mapping of a Building

SLAM is a technique used by robots to build up a map within an unknown environment (without *a priori* knowledge), or to update a map within a known environment (with *a priori* knowledge from a given map). Since our robots are equipped with simple hardware capabilities, the primary mapping technique will involve simple sonar and ultrasonic range finders to read distances as the mapping takes place.

4.2 Human Rescue

In hazardous conditions when it's too dangerous to human rescuers to reach remote places, a swarm of robotic agents can be deployed to search and rescue The task of human/object rescue requires the robotic agents to cooperatively work together to pull a heavy object to a desired place. A dummy human object has been used to test different pulling scenarios.

4.3 Painting a Wall

The task's objective is to perform interior painting job using two robots each equipped with an arm that has a1-Dof gripper attached to a 2-Dof arm which controls the position of the end effecter allowing it to rotate up, down and a 360 degree rotation around its own center.

CHAPTER FIVE: EXPERIMENTS AND RESULTS

A new software deployment environment for heterogeneous robots has been presented in the previous chapter. In this chapter, we compare how the proposed system stands among existing systems. The proposed software environment utilizes robots that have different modular designs and configurations of sensory modules and actuators. The embedded middleware feature allows the robotic agents to extend their configuration by auto-detecting newly added components. The proposed solution successfully overcomes most of the limitations found in previous software environments. These can be summarized as follows:

1. The system defines a set of rules and constructs programs that make the different robotic agents work in a swarm fashion even though they are heterogeneous in their hardware configurations and functionalities.
2. Robustness against failing sensory/actuation components while the task is taking place is one of the core functionalities of the decentralization approach. The decentralized algorithm allows robots to reason, reassign, and execute their intended missions.
3. Another key feature of the new system is that it uses C# programs which utilize the strong C# built-in libraries to interface with vast types of microcontrollers.

Previous systems have a slightly different method to communicate with robots. For example the MAJIC software [8] sends only basic commands using its own programming language to the group of selected online robots connected to the different serial ports. MAJIC software does not upload complete programs to robotic agents to allow them work in a swarm fashion. The DCF [55] on the other hand, uploads the entire program to robotic agents and also it provides an online simulator. However, The MDLE language does not work with the recent robotic platforms. Moreover, the DCF system was particularly designed to be used on more sophisticated robotic system which posses more computing power. More precisely, robotic agents with onboard laptops were used to execute the programs the DCF generates.

4. Unlike the player/stage system software, UBSwarm is not a real-time simulator, although it can monitor live data being sent by the multiple robots to the serial ports of a central computer. The player/stage is basically a simulator that reflects the movements and actions of the robotic agents as they perform their tasks in a graphical two-dimensional environment rendered on a computer screen.
5. Our system explains very clearly how programs are constructed and how functional blocks of code are being fetched as the users feed their inputs to the system interface. Certain object-oriented classes are added to the program that incorporates the specific task/application in real time.
6. Unlike the other systems, both decentralized/centralized coordination modes are adopted in UBSwarm. UBSwarm also integrates sophisticated programs that provide communication based coordination between the robotic agents.

A comprehensive attribute-based bibliography which compares our proposed system with the current robot deployment systems is given in table 5-1.

Table 5-1: Attribute-based comparison between the proposed system and the previous environments

Name	**System Model**	**Control Model**	**Fault tolerance**	**Real-time**	**Distributed environment**	**Simulator**	**Standards and Technologies**
E-Puck	Layered architecture; server/client; OS platform independent	Service-oriented architecture; no formal language	Yes	No	Yes	No	Local procedure calls (RPC) to call functions. J2EE is used
UBSwarm	Component-based framework; I/O Master/slave communication. Platform independent model (PIM)	Service-based procedure calls. Runtime agent; dynamic function calls	Yes. Embedded compiler; Robots have redundancy	No, But Can be added	Yes, services installed on server can be called on another machine	No	C# libraries; python and C++ programs; The interface is built in JAVA
DCF	Component-based architecture	XML file stores information needed to communicate	Yes	Yes	No	Yes, Limited to some apps	Needs Java virtual machine; uses special language called MDLE
MAJIC	Client/Server; component-based framework;	Client/server; centralized control	No	Yes	No	No	TCP protocol, Direct procedure calls. Java scripts uploaded to robots
Pyro	Architecture independent		Yes	No	Yes	No	Socket based using TCP protocol, XML, SOAP, OpenGL, HTTP
Player/ Stage	Client/server; decentralized control	Centralized model. Networked interface	Yes	Yes	Yes	Yes, 2D and 3D	3-Tier Architecture based on proxy objects.
Mobile-R	Component-based architecture	Offline and online dynamic task allocation, Neural Network (ANNs)	No	Yes	Yes	No	IEEE Mobile-C library Generic algorithms (GA)

5.1 Simultaneous Localization and Mapping (SLAM)

As illustrated in table 3-1, robot teams are composed of heterogeneous types of robots. A maximum of three robots were used in the SLAM experiment where R1,R2 are equipped with a laser-scanner and a v32 sonar scanner mounted on servo motor that rotates 180 degrees, and a camera mounted on the front used for object recognition. R3 is equipped with a sonar scanner mounted on a servo motor only. To accomplish the task, robots must navigate from a starting position and stops when a base station that runs our SLAM program generates a complete map of the building.

Each robot is placed randomly in the building to be mapped. The robots start scanning the surrounding area by moving forward while constantly maintaining 30 cm from the wall on its left side. An ultrasonic range sensor mounted on the top of each robot will turn 45 degrees to the right, it scans, and then it turns another 90 degrees to read all distances from the wall or the other obstacles. The scanner then rotates to the center position, it scans and then it turns to the left side as the ultrasonic sensor turns 90 degrees twice to the left. The process will be repeated every 30cm until it gets to the far side of the building. Encoders on each robot's wheels measure the distance the robot has covered as it scans. These two readings may be combined with a third reading from sonar sensors mounted on each side of the robot to add more accuracy and redundancy to the scanning ability. All together, those readings generate two-dimensional values that are fed to a Matlab program on a base station which in turn generates a 2-D map of the scanned area. Each robot communicates with the base station using Wireless Xbee modules, which provide communication via Wireless Wi-Fi 802.11 b/g/. One Xbee module is attached to

the base computer through USB port. As far the Matlab program is concerned, the SLAM algorithm uses the well-known Extended Kalman Filter (EKF) to predict and refine measurements. The Extended Kalman Filter (EKF) is an updated nonlinear version of the Kalman filter which relies on the current mean and covariance to predict an estimate. In the extended Kalman filter, the state transition and observation models are differentiable functions.

$$X_k = f(X_{k-1}, U_{k-1}) + W_{k-1} \tag{5.1}$$

$$Z_k = h(x_k) + V_k \tag{5.2}$$

Where W_k and V_k are the process and observation noises, X_k is the state vector and Z_k is the observation vector. The functions *f*() and *h*() are process and observation nonlinear vector functions respectively. The MATALB SLAM interface is a modified version of an open source program developed by Jai Juneja [67]. The software has been upgraded to make it run in real time by receiving live measurement data from the onboard sensors and wheel encoders. The software is also modified to simulate two or more robots in an attempt to meet our experimental objective. The SLAM program receives readings from each robot's ultrasonic range finders, wheel encoders and/or sonar readings. The software takes as an input a vector of readings from the two motion estimates received from sensor scanning and wheel odometers on each robot.

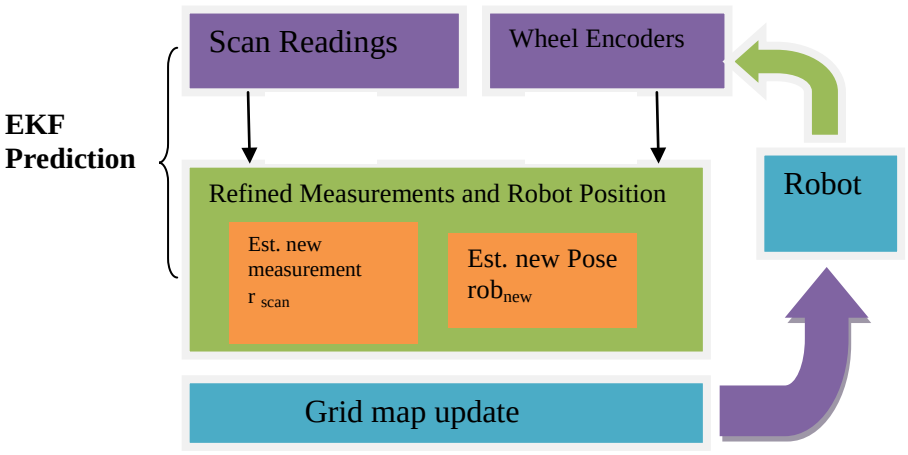


Figure 5-1: Multi-robot mapping using EKF prediction

The refined pose estimate is the result of the EKF processing of the two estimates, which takes into account their relative uncertainties. Figure 5-1 summarizes the new EKF technique.

5.1.1 Communication

Communication with the host computer is essential in this experiment. The robot must maintain the communication link to the host computer at all times. Transmission of data (X_1, X_2, X_3, *and* Y) from the swarm robots is transmitted at the end of each scanning for every target distance of twenty centimeter. A special command is sent from the base station to initiate internal clock which enables a robot to transmit its data once at each clock cycle as illustrated in figure 5-2. Communication with the base station is accomplished according the following algorithm:

Enable system clock each cycle 500 milliseconds

1st clock cycle enable transmission from robot 1

2nd clock cycle enable transmission from robot 2

3rd clock cycle enable transmission from robot 3

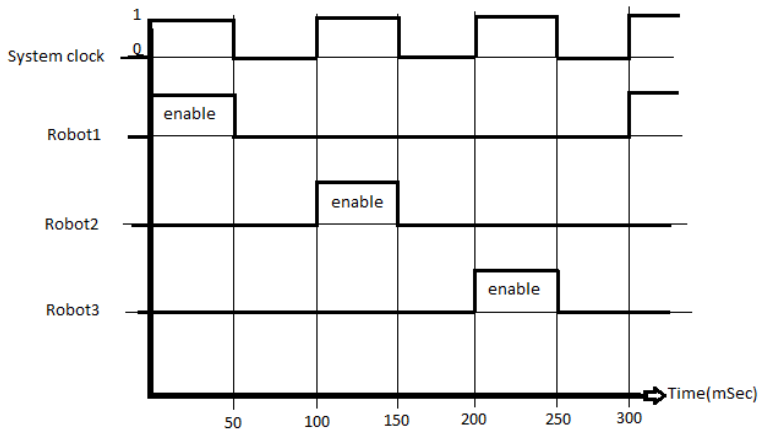


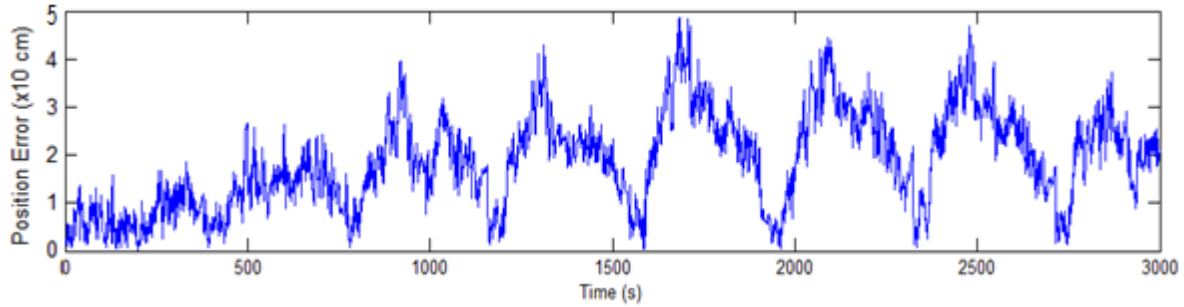
Figure 5-2: Data transmission clock from each robot to the base station

We performed three successful trials for each experiment set by varying the number of robots in each experiment. The mapping program is uploaded to each robot using two kinds of configurations that are set in the UBSwarm environment interface as follows:

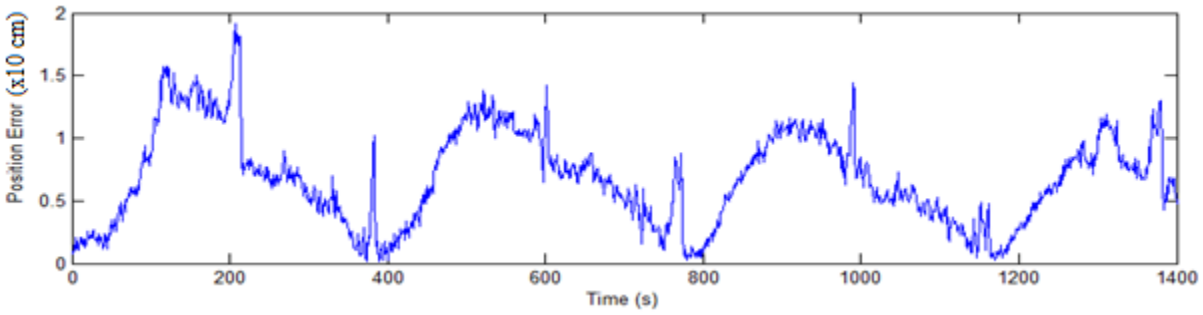
1- Wheel encoders, one sonar sensor mounted on the front, and one onboard ultrasonic range finder.
2- Second configuration uses the same settings as the above plus two more sonar sensors mounted on both sides.

The first experiment deploys one robot (equipped with two sensing components and both are used for obstacle avoidance). The second experiment deploys two robots and also they are equipped with two sensing components whereas three robots each equipped

with three sensing components were deployed in the third experiment. In the first experiment, the mapping task was completed in about 33 minutes. In the second experiment, the task took 16 minutes to complete, whereas it took 10 minutes to complete in the third experiment.

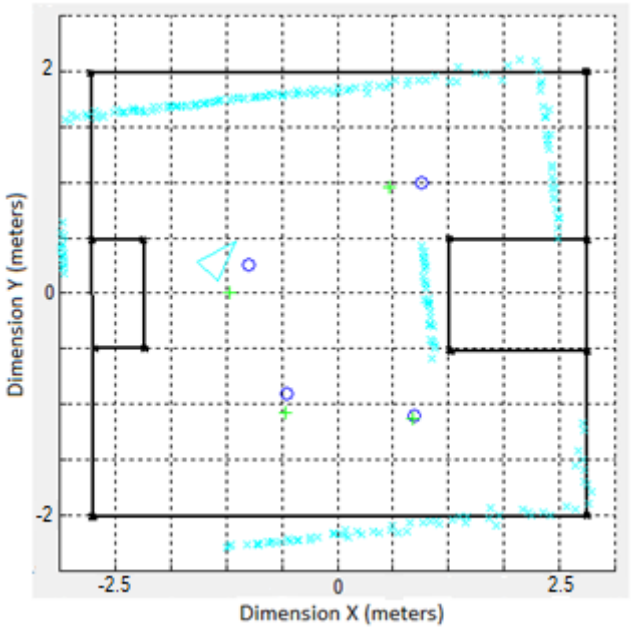


(a)

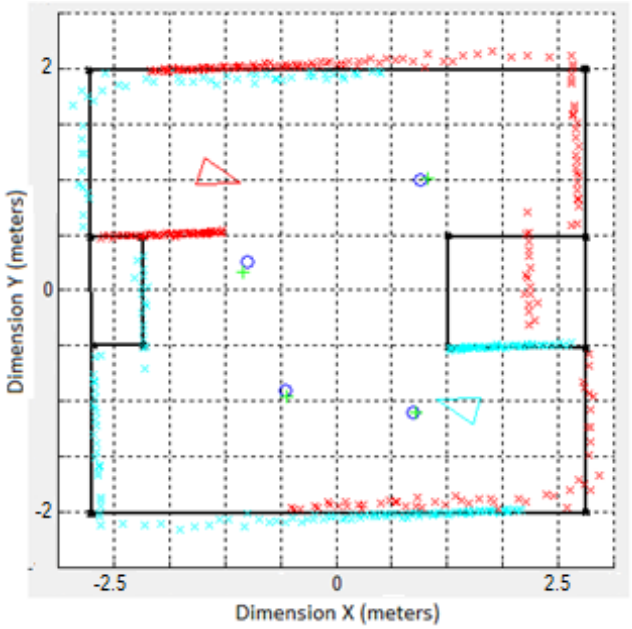


(b)

Figure 5-3: (a) Scan error during runtime using one robot, (b) scan error during runtime using two robots

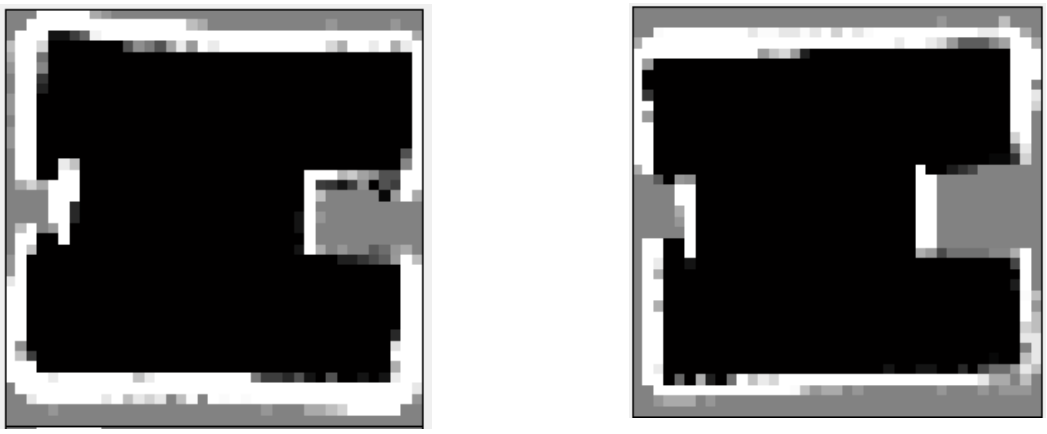


(a)



(b)

Figure 5-4: (a) Position of one robot and its scanned estimates vs. actual map, (b) Two robots positions and scanned estimates versus actual map



(a) (b)

Figure 5-5: (a) the map generated by one robot, (b) the map generated by the two robots.

Figure 5-3 (a) and (b) shows the measurement error generated by the one and two robots as opposed to the actual positions of walls. The region these robots are trying to map is a 5x4 square meters classroom with two tables placed at the shown locations. The algorithm decides the number of robots needed based on the dimensions of the area it is going to map which in this case it was two robots. It can be seen clearly that the map generated by two robots is more accurate than that generated by one robot.

Figure 5-4 (a) and (b) shows the actual map (black outline) and the estimated measurements (blue and red dots) generated by one and two robots respectively (blue and red triangles). Figure 5-5 (a) and (b) shows the rendered map generated by one and two robots respectively. We notice that more accurate white outline has been generated using the latter experiment. Figures 5-6 and 5-7 show the results of the third experiment that is when three robots are used (red, green and blue triangles). A different type of a range sensor is used on each of the three robots in the third experiment. Please note that the

maximum position error generated by one and two robots were 50 and 20 centimeters as shown in figure 5-3 (a) and 5-3 (b) respectively, while the maximum position error was 15 centimeters in the third experiment as shown in figure 5-7 (b). The average is taken between the two readings (on board ultrasonic sensor and side sonar sensors). Such addition will boost the accuracy of the measurements as well as adding redundancy to the robotic system should any sensor fails when tasks are being executed.

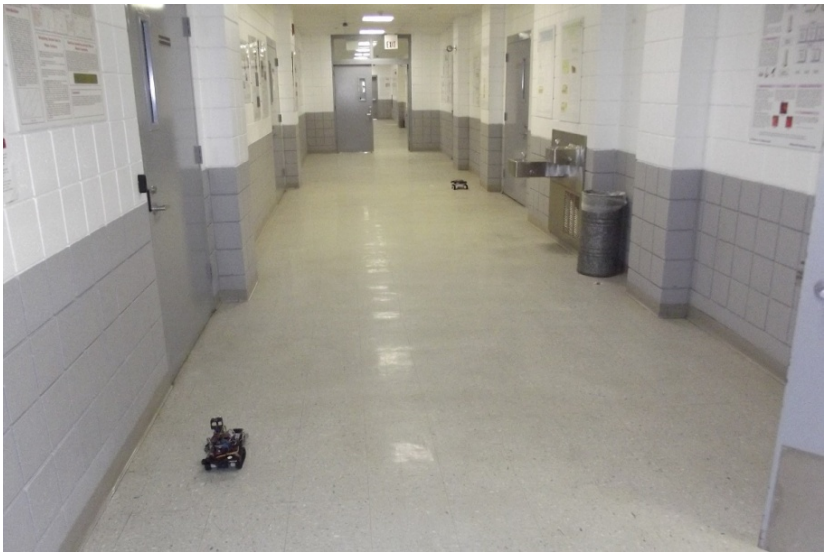
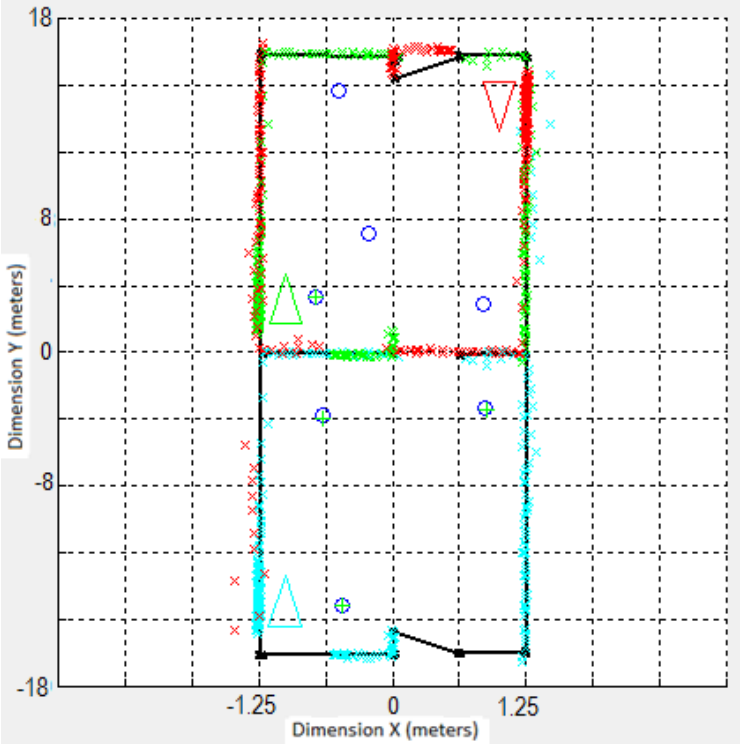
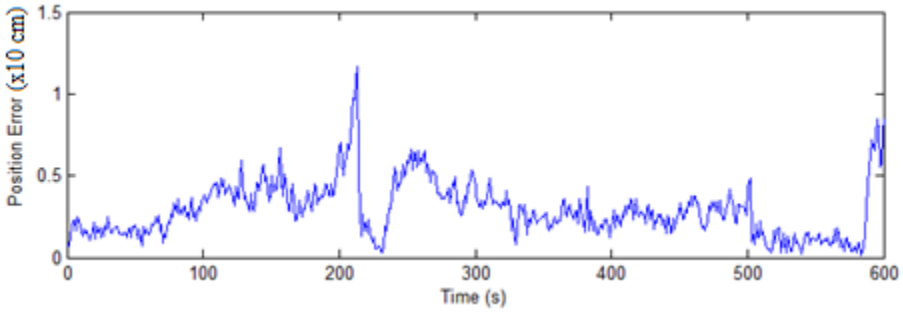


Figure 5-6: Three robots performing mapping



(a)



(b)

Figure 5-7: Experiment three (a) The estimates generated (b) Position error (centimeters) in 10 minutes of runtime

5.2 Human Rescue Task

The human rescue algorithm has been developed forUBSwarm so that robots can autonomously cooperate and coordinate their actions so that a human dummycan be pulled away in a minimal time. Cooperation between robots is achieved by exchanging

messages when an additional robot is needed to pull the object. First, the software environment deploys a particular type of robot that searches for a human dummy as it wanders in the unknown environment; such a robot is equipped with onboard camera allowing it to detect a white stripe attached to the human body lying on the ground. Video frames are received at a base station computer. The frames are fed to Matlab program that detects the white stripe using a line detection module as shown in figure 5-8. The algorithm incorporates Hough transform and enhanced edge detection algorithms.

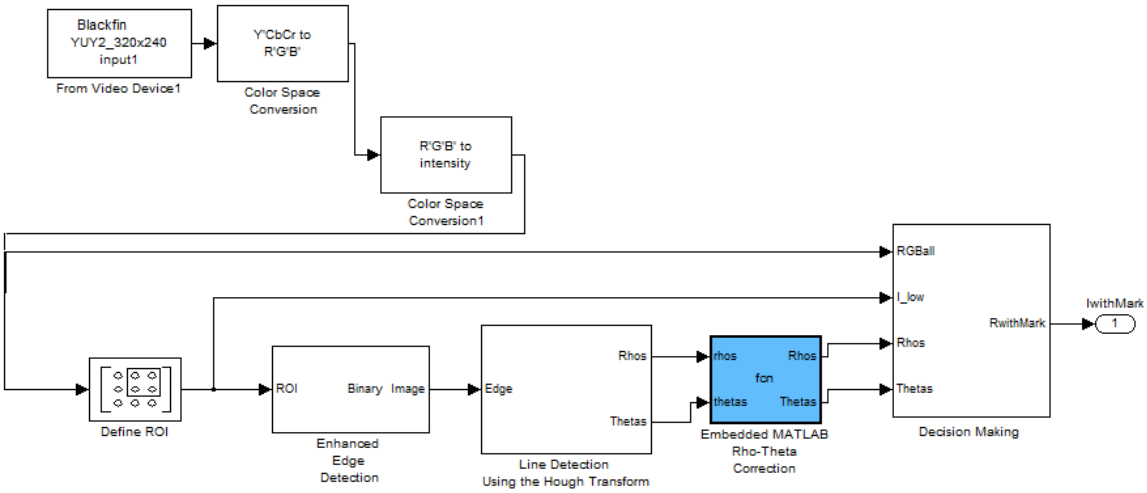


Figure 5-8: Overview of line detection Module

If more robots are needed to pull the object, the robot calls another agent using Xbee-based communication module. Wheel encoders on each robot are used to decide whether or not to call more robots. When the pulling subtask is being performed by a robot, its wheel encoders read the elapsed distance. If the distance is zero, it calls for more agents to be sent. Robots place themselves at different locations. Using their grippers and by sending a special synchronization message, the robots attach themselves

to the body and start pulling backward towards the goal position. A human prototype was built and several experiments were conducted. As the weight of the human increases more robotic swarm agents were called. We noticed that the configuration that uses more than three robots is able to successfully pull the object. However; this configuration may cause the robots to skid to any side. Consequently, this act increases the time taken by the robots to complete the task. Dispatching the right number of robots is the goal solution that is generated by the algorithm embedded in UBSwarm. Figure 5-8 shows a human-like dummy being pulled by four robots. We ran centralized as well as decentralized UBSwarm modes by performing three trials for each experiment set indicated by the number of robots, and obtained data on the completion time and the number of successful experiments. In total, we have performed 24 trials.

In the last experiment set, when four robots were used, we triggered faulty sensors at time 100 second to illustrate the fault-recovering capabilities of swarm team. In that experimental set, R5 performs its assigned tasks according to the plan. During the execution, the camera on R5 is covered in a way that it cannot detect the object. Eliminating this sensor triggers the coordination manager on the centralized station to generate new solutions for the rest of the team (three robots) to accomplish the task. In the decentralized approach, robots are always in one of the following states: reasoning, auctioning, navigating, and idle. A robot starts reasoning when it receives a task announcement. We introduced the same kind of failure as that of the centralized approach. In this example, at time 100 seconds, all robots receive the task announcement of pulling and start reasoning to calculate utilities. At time 101 seconds, utilities are

calculated, and robots start to bid for the task and wait for the response. At time 105 seconds, the task is assigned to the rest of the team and then the robots continue their interrupted task. The least time successful solution to transporting task is found using a robot team that is constructed of three robots; R1,R3, and R4. This result is obtained using the RUTA algorithm embedded in the coordination manager component. This team was able to accomplish the transporting task in an average of 201 seconds using the centralized approach. The transporting experiment was conducted using decentralized approach as well. In this experiment, the decentralized parameters such as the negotiation-timeout value were set as follows: wait for reply is 0.85s. The team that is constructed of the same three of robots as of that in the centralized approach also had the minimum time to complete the task at an average of 277 seconds. Table 5-2 shows performance data collected from centralized experiments.

Table 5-2: Successful pulling distance according to different number of robotic agents

Team Size	**Weight of body**	**Average Pulling distance (meters)**	**Average Time (seconds)**
1	300g	1.6	196
2	800g	1.3	240
3	1200g	2.5	201
4	1200g	2.0	210
5	1200g	1.6	400

As an example, in both approaches the total cost of task (T_{rescue}) performed by the robots $r_i's$ in the capability-based ordering (R2, R3, R1, R4, R5) is determined by the robots utility functions associated with each of the following tasks:

$$T_{rescue} = \sum_{i=1}^{5} U_{rescue(i)}$$

$$\sum_{i=1}^{5} U_{rescue(i)} = \sum_{i=1}^{5} (utility_{i(nav)} + utility_{i(detect)} + utility_{i(grip)} + utility_{i(pull)})$$

$utility_{ij} = \max(0,\ u_2 \times (d_{ij}{}^{-1/2} \times \varphi_{given\ ij} \times pri_{ij})$, Where j = 1,2,3,4 i = 1,2,3,4,5, $U_{rescue(i)}$ is the overall utility of robot r_i, and $utility_{i(nav)}$, $utility_{i(detect)}$, $utility_{i(grip)}$, $utility_{i(pull)}$ are the navigation, object detection, gripping, and pulling subtasks.

5.2.1 Simulation Module

Deploying the right number of robots to rescue a human cannot be determined unless the weight of the human is known prior to running the experiment. However, running a premature simulation that has prior knowledge about experiment will provide a clear picture whether or not the robots are able to accomplish the task. The prior knowledge includes the human weight, its distance from the robots, and any other essential parameters (such as robots' wheel slippage percentages). The simulator calculates utilities of the robots and shows their behavior in a 3D motion. To illustrate that, the utilities are calculated for the different sub-tasks in the three-robot team. The first subtask to be performed is navigation; the utilities for the three robots (1,2, and 4) using the centralized approach are calculated as follows:

Robot 1, j = navigation

$$utility_{1(nav)} = \max(0, \quad u_2 \times (d_{1j}{}^{-1/2} \times \varphi_{given\ 1j} \times pri_{1j})$$

$$\varphi_{given\ 1j} = \varphi_{nav\ 1j} = 0.7\ \left[\left[\frac{prem_1}{rate_{servo\ (1)}}\right] \times \frac{1}{w_1}\right]$$

$$\varphi_{given\ 1j} = 0.7\ \left[\left[\frac{2200}{130}\right] \times \frac{1}{3}\right]$$

$$\varphi_{given\ 1j} = 0.7\ [5.58]$$

$$\varphi_{given\ 1j} = 3.90$$

Initially priorities of all sub-tasks are equal to 1, and $u_2 = 1$ hence,

$$utility_{1(nav)} = \max(0, \quad 1 \times (1^{-1/2} \times 3.90 \times 1)$$

$$utility_{1(nav)} = 3.90$$

Robot 2, j = navigation

$$\varphi_{given\ 2j} = \varphi_{nav\ 2j} = 0.7\ \left[\left[\frac{prem_2}{rate_{servo\ (2)}}\right] \times \frac{1}{w_2}\right]$$

$$\varphi_{given\ 2j} = 0.7\ \left[\left[\frac{2200}{320}\right] \times \frac{1}{1}\right]$$

$$\varphi_{given\ 2j} = 0.7\ [6.87]$$

$$\varphi_{given\ 2j} = 4.81$$

So,

$$utility_{2(nav)} = 4.81$$

After calculating for R4,

$$utility_{4(nav)} = 1.21$$

At time 110s, the gripping subtask was already scheduled at time 20s, the utility values for robots 2 and 4 are:

R2, j = grip

$$\varphi_{given\ 2j} = \varphi_{manip\ 2j} + \varphi_{sens\ 2j}$$

$$\varphi_{manip\ 2j} = 0.7\left[\left(\frac{t_{sj}}{t'_m}\right)\left[\frac{Prem_2}{rate_{act\ 2}}\right]\right]$$

$$= 0.7\left[\left(\frac{9}{200}\right)\left[\frac{2000}{60}\right]\right] = 1.05$$

$$\varphi_{sens\ 2j} = 0.9\left[\left(\frac{t_{sj}}{t'_m}\right)\left[\frac{Prem_2}{rate_{sens\ (2)}}\right]\right]$$

$$\varphi_{sens\ 2j} = 0.9\left[\left(\frac{11}{200}\right)\left[\frac{2150}{65}\right]\right] = 1.64$$

$$\varphi_{given\ 2j} = \varphi_{manip\ 2j} + \varphi_{sens\ 2j} = 2.69$$

$$pri_{2j} = \frac{1}{2} \times max[(u_1 \times (110 - 20), 0]$$

$$u_1 = 0.01$$

$$pri_{2j} = 0.45$$

Assuming robot 2 distance to the object is 3 meters

$$utility_{2(grip)} = \max(0, \quad 1 \times (3.0^{-1/2} \times 2.69 \times 0.45) = 0.69$$

The same applies to robot 4. Its corresponding utility values were

$$utility_{4(grip)} = \max(0, \quad 1 \times (4.3^{-1/2} \times 5.04 \times 0.45) = 1.09$$

Figure 5-9 shows a simulated 4 robotic agents performing the task of pulling a 1200g body.

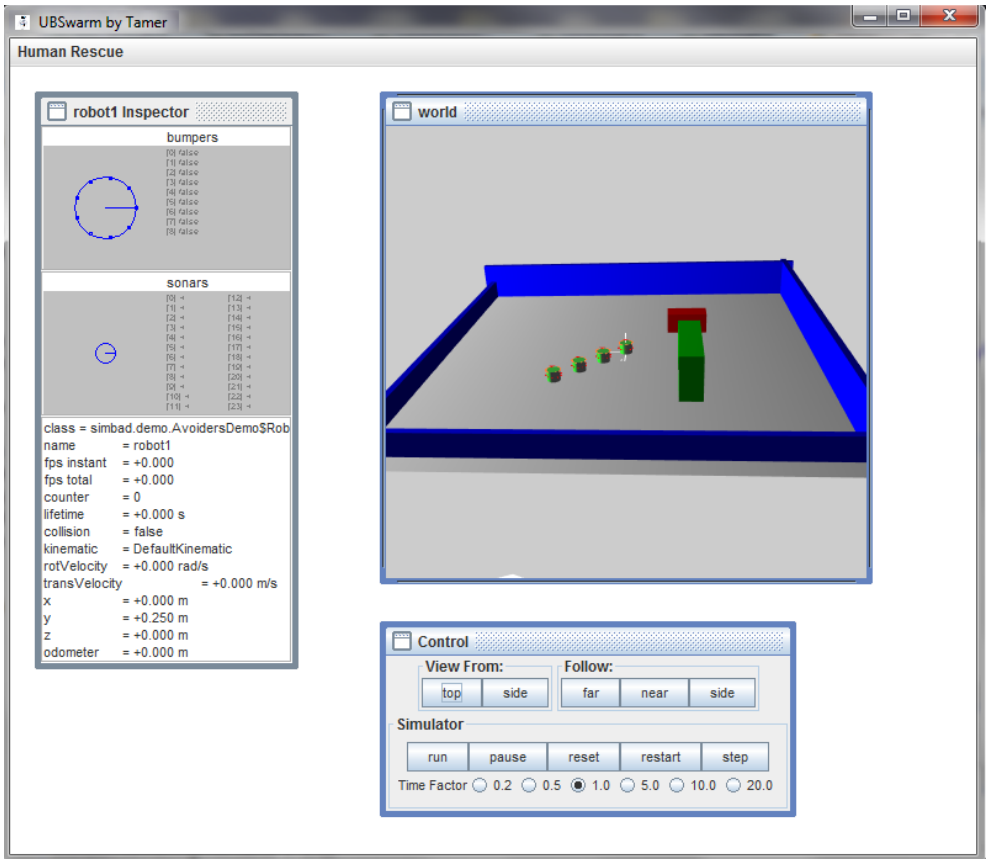


Figure 5-9: Four robots simulated before being deployed

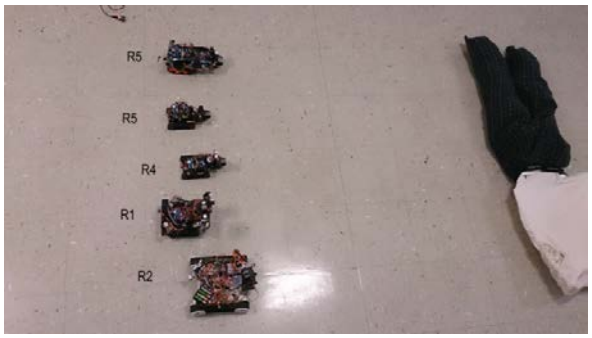
When evaluating the performance of two versus N robots, each team's utility value is the key factor that distinguishes the least time solution among the various team sizes and compositions. At the beginning, each team's utility is calculated as an initialization step in the RUTA algorithm. At this stage, the larger the team the higher utility value is. However, some team utilities might start to decline depending on their parameters as the task is taking place. The team that sustains high utility value throughout the course of performing the task until its completion will determine the minimum execution time and hence the optimal solution. Table 5-3 shows the order of teams'

success based on their utility values and their completion time. Please note the higher team utility the more successful the experiment is.

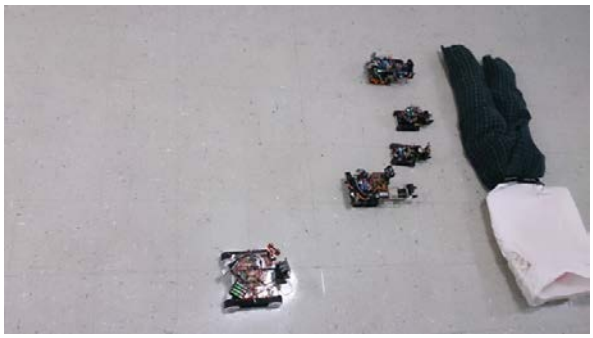
Table 5-3: Centralized vs. Decentralized team utilities

Team composition	Centralized		Decentralized	
	Utility value	Time (sec)	Utility value	Time (sec)
(R1,R3,R4)	9.63	201	6.91	277
(R1,R3,R4,R5)	8.82	210	6.62	299
(R2,R3,R4,R1,R5)	8.43	400	6.66	405
(R2,R5)	8.16	240	6.34	310

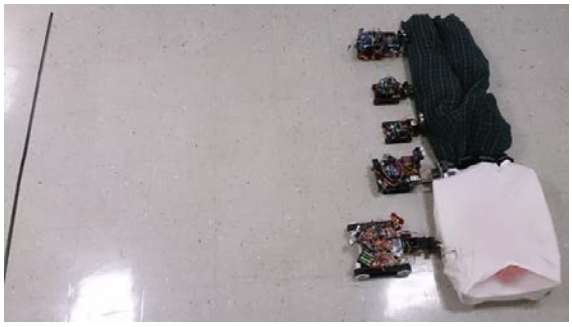
The desired pulling distance for 1200g human dummy was 2.5 meters. The sequenced photos in figure 5-10 below show an example of five robots pulling the dummy.



(a) Five robots are configured for a rescue



(b) Four robots are approaching



(c) The fifth robot, R2 is called

(d) Five robots crossing the finish

Figure 5-10: A dummy being pulled for 2.5 meters using five robots

5.3 Wall Painting

The task is executing an interior painting job. The robotic agents each equipped with location sensors, simple communication modules, and vision capability are able to paint their designated part of the wall. Painting using a brush is the most commonly used by human workers. Using a brush requires more sophisticated robotic arms. Painting using a spray, however, is less demanding in terms of accuracy and therefore more appropriate for our robotic agents.

5.3.1 Arm and End-effecter

The end effecter is basically a 1-DoF gripper attached to a 2-DoF arm which controls the position of the end effecter allowing it to rotate up, down and a 360° rotation around its own center. Figure 5-11 shows the movements and the offsets along the direct Z axis.

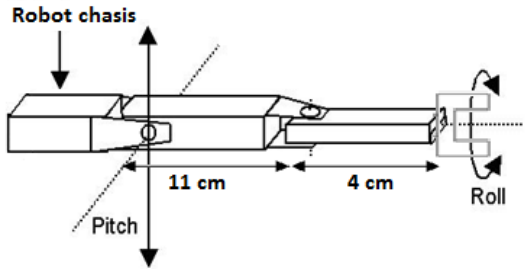


Figure 5-11: The 2-Dof sketch for the robot manipulator

Performing a painting job using this particular type of end-effecter creates multiple adjacent rectangular coating sectors as shown in figure 5-12. The height of each sector (H) is the height of the highest point the end effecter can reach on the wall which

was reasonably taken as 30 cm. The width of the sector (W), for a robot with a given work space), depends on the area being sprayed by the nozzle attached to the end effecter (gripper), that actually determines the width of a stripe (S) painted in a single tool movement.

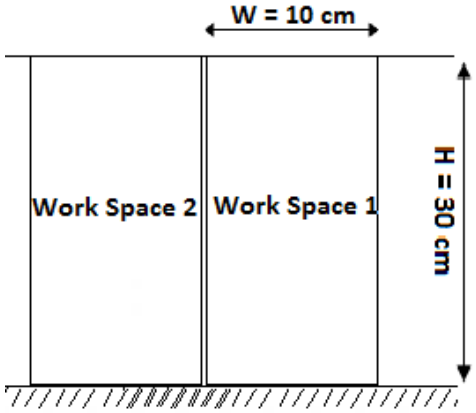


Figure 5-12: The surface covered by the painter

Figure 5-13 shows two robots performing a painting test and the nozzles attached to each of their grippers.

Figure 5-13: Spraying nozzles attached to the Robots

5.3.2 Painting Method

The two robots were used are equipped with a 2-Dof manipulator and a flexible hose attached to the end effecter on one end and to a compressed paint container on the other end. Using this flexible spraying equipment, each robot can paint a surface of 10 x 30 cm only when it is facing the surface of the wall. The trajectory of the end-effecter is composed of three kinds of movements:

1. Each robot moves concurrently at the same speed as that of the other robots. An infrared sensor mounted at the front of each robot. When a wall is detected, each robot will rotate its painting tool in order to align it with the wall at the highest extension then it will maintain a constant distance from the wall as the painting starts.
2. The tool will be moved in two linear vertical movements in which the paint is being sprayed. During the movement, each sprayer is activated or de-activated according to the distance from the wall.
3. After completing two vertical sprays, each robot will move to the next adjacent partition on the wall by moving backward for a predetermined fixed distance, turning to the a left, moving forward for 20 cm, turning to the right and then moving forward for the same fixed distance to reach the next partition. The painting process takes place again and the whole procedure is repeated until the whole area is painted.

We are interested in learning how much time is saved when painting using multiple robots. To do so, we ran two experiments. The first experiment which involved

one single robot, the task was completed in 30 minutes. In the second experiment the task was completed in 14 minutes using two robots.

5.4 Performance of Centralized vs. Decentralized Approaches

When it comes to evaluating the performance of both of the centralized and decentralized approaches, we observe that the centralized approach performs quicker with more flexibility while generating plans. Little time is needed to initiate new ordering of the robot team since any change in the capabilities of the team needs only to be updated locally. The centralized knowledge base also needs to be updated when the team capabilities change. However, the centralized approach takes a little longer time to find a solution than that of decentralized approach. Decentralized UBSwarm runs on each robot. The solution can be found using less time. However, except for single-robot team, this method trades off solution quality because of the less computational power of the robots when compared to the base station computer. To increase robustness against sensory failures, the decentralized approach on every robot. Robots share capability information with each other at the beginning or whenever the team capabilities change. This method requires more work to maintain the knowledge base than the centralized approach on a single base station, since the knowledge base updates must be duplicated on all robots.

In centralized UBSwarm, the total time for generating a solution is the time to assign subtasks (m) to the current team ordering which increases exponentially O(nm), whereas the time needed for the decentralized approach is the auctioning time which is O(1) plus the time taken by each robots to respond which is O(n). Here n is the number of

working (unfailing) robots. As shown in Fig. 5-14, the average time to generate a solution increases as the robot group size increases, linearly for Decentralized UBSwarm, and exponentially for centralized UBSwarm. Additionally, in Fig. 5-15, the plan utility is plotted for four different team sizes. At time 100 seconds, an error is introduced to one of the robots in each team. We can observe that the team accumulated utility drops down at that point then as both approaches re-allocate the tasks the overall utility increases. The figure shows the accumulative team utility over time. The sub-task that is assigned to the faulty robots is taken over by the rest of the team as a result of the reasoning algorithms executed by the two control schemes. If no faults occur during execution, team utility should maintain a slight decrease in their values. Additionally, the centralized results always have a higher utility value than that of the decentralized approach, because the centralized approach operates with complete information received from the robot team. Moreover, the decentralized approach's core functionality is based on the use of time-based parameters (i.e. wait-for-reply t_{out}) that not only requires more communication overhead amongst the robots but also increases the time slot given to the particular sub-task and thus increases the execution time.

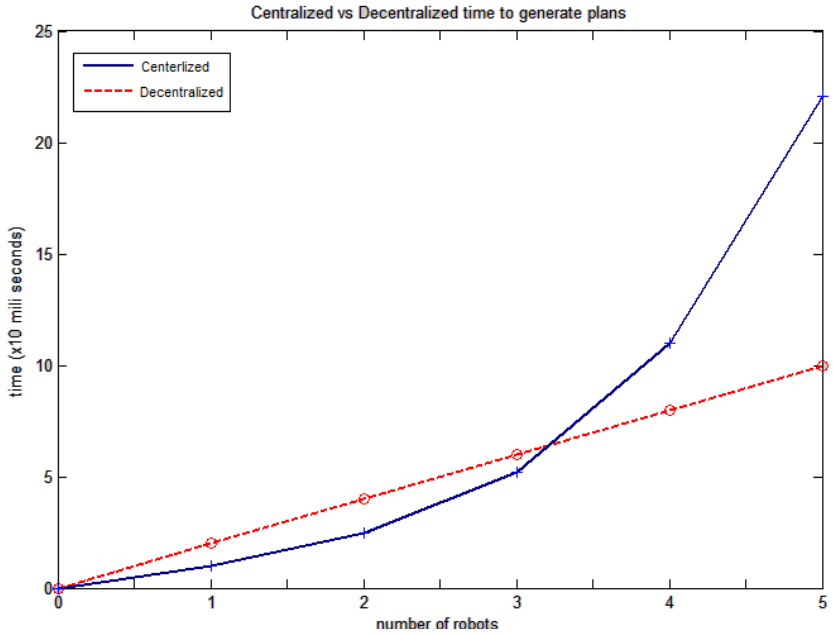


Figure 5-14: Centralized vs. Decentralized time needed to generate solutions

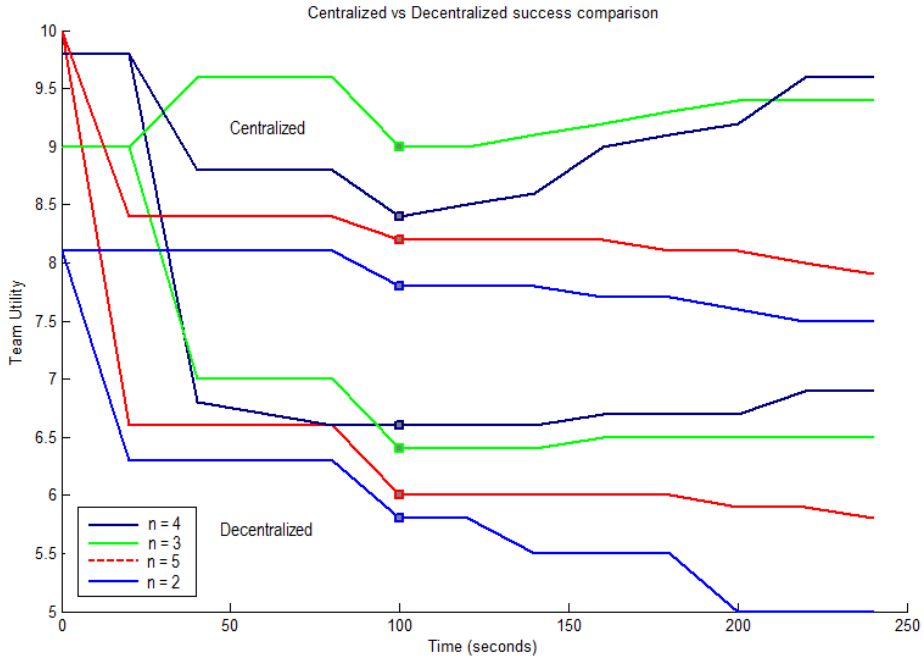


Figure 5-15: Centralized vs. Decentralized team utility

Based on the task requirement, the centralized algorithm first picks at least two robots which calculate the two highest utility values. The system then assigns the first subtask according to the task schedule as an initialization step. Once the two robots are deployed, the coordination manager component takes over the control. Based on the values of the two robot's utilities or the addition of new robots (or removing a faulty robots from the solution plan), the coordination component assigns or reassigns subtasks to the robotic team. When decentralized approach is used, the problem of reconfiguring solutions is left to the robots themselves to resolve. Figure 5-15 shows the effect of a faulty sensor introduced on each of the different team sizes (N = 2,3,4,5) in the human rescue task simulating a dummy that weighs 1200gms. We notice that the combination of robot teams which gives an optimal solution is the combination of the three robots (R1, R2, and R4). This specific task requires at least one robot equipped with a camera for the purpose of detecting the dummy object. Table 5-4 below shows the highest utility value of each of the different team combinations for centralized and decentralized approaches at time 225 seconds.

Table 5-4: Team compositions and their utility values

Team composition	**Utility values at time 225s**	
	Centralized	**Decentralized**
(R1,R3,R4)	9.63	6.91
(R2,R4)	9.40	6.53
(R2,R3,R4,R1)	8.68	6.42
R2	7.80	4.98

5.5 Comparison between RUTA and Current Techniques

The above experiments present the results of applying UBSwarm to various multirobot applications and the robustness of the UBSwarm-Decenralized approach. It is worth mentioning how our proposed approach to the OAP stands amongst the existing techniques by comparing their scalabilities and execution time (for n robots and m tasks) to our approach. According to table 2-1 in our literature survey, approaches to robot task allocation are divided into behavior-based and market-based approaches ALLIANCE is a behavior-based technique in which each robot performs a greedy task-selection algorithm for each task yielding a O(mn) per iteration where m and n are the number of tasks and robots respectively. At each iteration, each robot compares its own utility to that of the other robots and selects the task for which it is capable to perform. Because robots have to share their utilities in each iteration, communication overhead of O(n) is added to the overall execution time. ACO-based task allocation [69] is another behavior-based approach. In this technique, each robot has a corresponding task utility that decides if the robot is capable of executing a task by estimating the robot's utility for that task. Utilities are computed in a task-specific manner as a function of relevant sensor data. These utilities are periodically broadcasted to the other robots simultaneously to allow reassignment of tasks. Since each robot must broadcast its utility for each task, the system has a communication overhead of $O(mn)$ per iteration.

Moving to auction-based approaches, In the M+ system, each robot considers all the currently available tasks at each iteration. For each task, each robot uses a planner to

compute its utility and announces the resulting value to the other robots. With each robot broadcasting its utility for each task, we have communication overhead of $O(mn)$ per iteration. Similar to M+, the MURDOCH task allocation mechanism also employs a variant of CNP. For each task auction, each available robot broadcasts its bid (i.e., utility), yielding communication overhead of $O(n)$ per iteration because of the asymmetric nature of MURDOCH's auctions. In ASyMTRe approach, the solution is based on perceptual schema representation of each robot's physical components. The solution requires the time to generate all the orderings of robots, which increases exponentially O(n!), and the actual reasoning time $O(mn^2)$ when utilities are being calculated. In ASyMTRe-D, the time is the average reasoning time O(mn) for the group to generate a solution.

We notice that our algorithm's execution time does not differ from that of existing homogenous approaches. So far we have not shown any actual comparison of our technique with the systems that we have analyzed. Because all of the previous architectures execute some kind of greedy algorithm for task allocation, the solution quality of greedy optimization algorithms can be difficult to define. Evaluating each architecture depends strongly on the nature of the experiment. The input to the experiment is the set of robots, tasks, and the environment that they are operating in. However, by taking each of the previous system's utility equations and applying them in our centralized approach, would give a proper comparison between our system and the current systems. As shown in figure 5-16, the comparison is made by calculating the

utilities of the systems for 15 human rescue trials performed using 1200gms dummy on teams that are composed of 2, 3, 4, and 5 robots respectively.

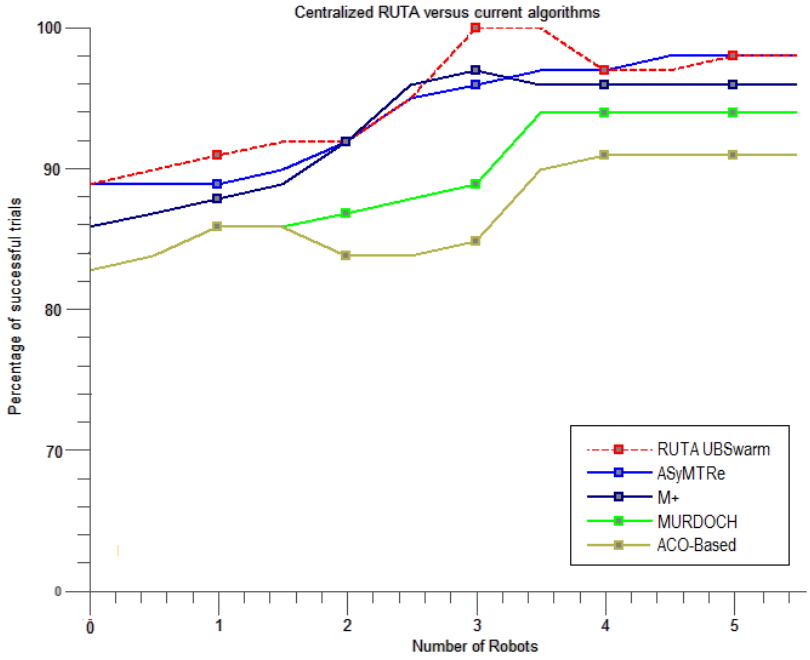


Figure 5-16: Comparison of RUTA with current methods

CONCLUSIONS AND FUTURE WORK

In conclusion, the creation of UBSwarm is specifically designed to deploy heterogeneous robotic agents. Based on the type, the number of the robotic agents available, and the task selected, UBSwarm automatically constructs optimal solutions to the three different missions specified by the user. The coordination algorithm is translated into programs customized for each heterogeneous robotic agent. These programs define a set of rules and behaviors that allow the different robotic agents to work in a swarm fashion, even though they have different hardware configurations.

Our work presents a task-oriented software application that facilitates the rapid deployment of multiple robotic agents. The task solutions are created at run-time and executed in a centralized or decentralized fashion by the robotic agents. A core component of the system's framework is responsible for generating these task solutions. At the robots' deployment and throughout their operational time, the software reconfigures solutions to accommodate any variation within the group of robots. Then the tasks are fractioned into smaller sub-tasks and assigned to the optimal number of robots using a novel Robot Utility Based Task Assignment (RUTA) algorithm. In addition, we demonstrated a reasoning algorithm that generates multi-robot utilities through a negotiation process in a decentralized manner.

Through the decentralization process, each robot generates an optimal solution for the entire task by reassigning subtasks to the team based on each robot's utility. The system has to account for any change in the number of robots being used. When comparing the centralized to the decentralized UBSwarm, we discover that decentralized UBSwarm provides more flexibility and fault-tolerance is that these solutions have less quality in forming solutions.

In the centralized situation, a set of robotic agents can adopt and adjust their subtasks in accordance with any variation that may occur. During runtime, the robot's status is shared between robots. If a failing robot interrupts a task, then the swarm robotic environment will reconfigure new task solutions in order to adapt to changes within the robotic team's composition. Analytical studies and physical implementations of coordination modes have been incorporated into our research.

In a broader view of the system, UBSwarm deployment environment reduces efforts in dispatching tasks to swarm robotic agents, and permits users to add various new functions for robots and sensors. A few of the future work improvements that can greatly enhance the decision making performed by the coordination component and its applications include:

1. Optimizing the control algorithm to decide shortest path in executing a task, locate object more accurately, shorten swarm intelligence decision time, and keep better power efficiency in the operation.
2. Another future improvement would be deciding the optimal number of robots to carry out the task most efficiently. Implementing error estimation on the fly

(during run-time) can be thought of, which can positively affect the decisions/configurations afterwards.

3. Developing algorithms leading to controlling nano robots for cancer cell detection/removal and implement neural networks to allow robots learn their environment as they navigate.
4. Improving the coordination algorithm using intelligent decision agent. Incorporate huge number of simpler robots (hundreds) to perform complex tasks.
5. Extending the functionality of the deployment environment to allow integrating more sensory and actuation devices

BIBILOGRAPHY

[1] Y. Xinan, L. Alei, and G. Haibing, "An algorithm for self-organized aggregation of swarm robotics using timer," in *Swarm Intelligence (SIS), 2011 IEEE Symposium on*, 2011, pp. 1-7.

[2] L. Bayindir and E. Sahin, "A review of studies in swarm robotics," *Turkish Journal of Electrical Engineering,* vol. 15, pp. 115-147, 2007.

[3] Z. B. D. Rus, K. Kotay, and M. Vona, "Self-reconfiguring robots," *Communications of the ACM,* vol. vol. 45, pp. 39-45, 2002.

[4] S. S. Nestinger and H. H. Cheng, "Mobile-R: A reconfigurable cooperative control platform for rapid deployment of multi-robot systems," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 2011, pp. 52-57.

[5] T. Fong, I. Nourbakhsh, and K. Dautenhahn, "A survey of socially interactive robots," *Robotics and autonomous systems,* vol. 42, pp. 143-166, 2003.

[6] K. Sugawara and T. Watanabe, "Swarming robots-foraging behavior of simple multirobot system," in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, 2002, pp. 2702-2707.

[7] H. Szu, P. Chanyagorn, W. Hwang, M. Paulin, and T. Yamakawa, "Collective and distributive swarm intelligence: evolutional biological survey," in *International Congress Series*, 2004, pp. 46-49.

[8] G. P. Ball, K. Squire, C. Martell, and M. T. Shing, "MAJIC: A Java application for controlling multiple, heterogeneous robotic agents," in *Rapid System Prototyping, 2008. RSP'08. The 19th IEEE/IFIP International Symposium on*, 2008, pp. 189-195.

[9] H. Dai, "Adaptive Control in Swarm Robotic Systems," *The Hilltop Review*, vol. 3, p. 7, 2011.

[10] A. Sayouti, H. Medromi, and F. Moutaouakil, "Autonomous and Intelligent Mobile Systems based on Multi-Agent Systems."

[11] R. Groß, M. Bonani, F. Mondada, and M. Dorigo, "Autonomous self-assembly in swarm-bots," *Robotics, IEEE Transactions on*, vol. 22, pp. 1115-1130, 2006.

[12] F. Mondada, G. C. Pettinaro, A. Guignard, I. W. Kwee, D. Floreano, J. L. Deneubourg, *et al.*, "SWARM-BOT: A new distributed robotic concept," *Autonomous Robots*, vol. 17, pp. 193-221, 2004.

[13] T. Fukuda and S. Nakagawa, "Dynamically reconfigurable robotic system," in *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*, 1988, pp. 1581-1586.

[14] Y. Mohan and S. Ponnambalam, "An extensive review of research in swarm robotics," in *Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on*, 2009, pp. 140-145.

[15] T. W. Dunbar and J. Esposito, "Artificial potential field controllers for robust communications in a network of swarm robots," in *System Theory, 2005. SSST'05. Proceedings of the Thirty-Seventh Southeastern Symposium on*, 2005, pp. 401-405.

[16] B. P. Gerkey and M. J. Mataric, "Sold!: Auction methods for multirobot coordination," *Robotics and Automation, IEEE Transactions on,* vol. 18, pp. 758-768, 2002.

[17] C. W. Reynolds, "Flocks herds and schools: A distributed behavioral model," *Computer Graphics*, pp. 25-34, July 1987.

[18] A. T. Hayes, A. Martinoli, and R. M. Goodman, "Swarm robotic odor localization," in *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, 2001, pp. 1073-1078.

[19] D. Payton, M. Daily, R. Estowski, M. Howard, and C. Lee, "Pheromone robotics," *Autonomous Robots,* vol. 11, pp. 319-324, 2001.

[20] R. Mayet, J. Roberz, T. Schmickl, and K. Crailsheim, "Antbots: A feasible visual emulation of pheromone trails for swarm robots," *Swarm Intelligence*, pp. 84-94, 2010.

[21] J. Svennebring and S. Koenig, "Building terrain-covering ant robots: A feasibility study," *Autonomous Robots*, vol. 16, pp. 313-332, 2004.

[22] R. A. Russell, "Ant trails-an example for robots to follow?," in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, 1999, pp. 2698-2703.

[23] R. A. Russell, "Heat trails as short-lived navigational markers for mobile robots," in *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, 1997, pp. 3534-3539.

[24] M. Mamei and F. Zambonelli, "Spreading pheromones in everyday environments through RFID technology," in *2nd IEEE Symposium on Swarm Intelligence*, 2005, pp. 281-288.

[25] H. Psaier and S. Dustdar, "A survey on self-healing systems: approaches and systems," *Computing*, vol. 91, pp. 43-73, 2011.

[26] V. Gazi and K. M. Passino, "Stability analysis of social foraging swarms," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 34, pp. 539-557, 2004.

[27] R. T. Vaughan, K. Støy, G. S. Sukhatme, and M. J. Mataric, "Blazing a trail: insect-inspired resource transportation by a robot team," in *Proceedings of the 5th International Symposium on Distributed Autonomous Robotic Systems (DARS), Knoxville, TN*, 2000, pp. 111-120.

[28] M. Blow, "'stigmergy': Biologically-inspired robotic art," in *Proceedings of the Symposium on Robotics, Mechatronics and Animatronics in the Creative and Entertainment Industries and Arts, The Society for the Study of Artificial Intelligence and the Simulation of Behaviour*, 2005.

[29] W. Liu, A. F. T. Winfield, J. Sa, J. Chen, and L. Dou, "Towards energy optimization: Emergent task allocation in a swarm of foraging robots," *Adaptive Behavior*, vol. 15, pp. 289-305, 2007.

[30] E. J. Barth, "A dynamic programming approach to robotic swarm navigation using relay markers," in *American Control Conference, 2003. Proceedings of the 2003*, 2003, pp. 5264-5269.

[31] N. R. Hoff, A. Sagoff, R. J. Wood, and R. Nagpal, "Two foraging algorithms for robot swarms using only local communication," in *Robotics and Biomimetics (ROBIO), 2010 IEEE International Conference on*, 2010, pp. 123-130.

[32] B. Wang, D. Hoang, I. Daiz, C. Okpala, and T. M. Sobh, "An Experimental Collective Intelligence Research Tool," in *proceedings of the Fourth International ICSC Symposium on Engineering in Intelligent Systems (EIS 2004), Madeira, Portugal*, 2004.

[33] J. McLurkin and J. Smith, "Distributed algorithms for dispersion in indoor environments using a swarm of autonomous mobile robots," *Distributed Autonomous Robotic Systems 6*, pp. 399-408, 2007.

[34] R. Simmons, S. Singh, D. Hershberger, J. Ramos, and T. Smith, "First results in the coordination of heterogeneous robots for large-scale assembly," *Experimental Robotics VII*, pp. 323-332, 2001.

[35] D. Vail and M. Veloso, "Multi-robot dynamic role assignment and coordination through shared potential fields," *Multi-Robot Systems*, pp. 87-98, 2003.

[36] E. Pagello, A. D'Angelo, and E. Menegatti, "Cooperation issues and distributed sensing for multirobot systems," *Proceedings of the IEEE*, vol. 94, pp. 1370-1383, 2006.

[37] S. C. Botelho and R. Alami, "M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement," in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, 1999, pp. 1234-1239.

[38] R. Zlot, A. Stentz, M. B. Dias, and S. Thayer, "Multi-robot exploration controlled by a market economy," in *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, 2002, pp. 3016-3023.

[39] M. Yim, W. M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, *et al.*, "Modular self-reconfigurable robot systems [grand challenges of robotics]," *Robotics & Automation Magazine, IEEE*, vol. 14, pp. 43-52, 2007.

[40] B. Salemi, M. Moll, and W. M. Shen, "SUPERBOT: A deployable, multi-functional, and modular self-reconfigurable robotic system," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, 2006, pp. 3636-3641.

[41] S. Kernbach, E. Meister, F. Schlachter, K. Jebens, M. Szymanski, J. Liedke, *et al.*, "Symbiotic robot organisms: REPLICATOR and SYMBRION projects," in *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*, 2008, pp. 62-69.

[42] A. Kushleyev, D. Mellinger, and V. Kumar, "Towards a swarm of agile micro quadrotors," in *Robotics: Science and Systems (RSS)*, 2012.

[43] F. Tang and L. E. Parker, "A complete methodology for generating multi-robot task solutions using asymtre-d and market-based task allocation," in *Robotics and Automation, 2007 IEEE International Conference on*, 2007, pp. 3351-3358.

[44] W. Burgard, M. Moors, C. Stachniss, and F. E. Schneider, "Coordinated multi-robot exploration," *Robotics, IEEE Transactions on*, vol. 21, pp. 376-386, 2005.

[45] V. Zykov, E. Mytilinaios, M. Desnoyer, and H. Lipson, "Evolved and designed self-reproducing modular robotics," *Robotics, IEEE Transactions on,* vol. 23, pp. 308-319, 2007.

[46] R. Moeckel, C. Jaquier, K. Drapel, E. Dittrich, A. Upegui, and A. J. Ijspeert, "Exploring adaptive locomotion with YaMoR, a novel autonomous modular robot with Bluetooth interface," *Industrial Robot: An International Journal,* vol. 33, pp. 285-290, 2006.

[47] S. C. Goldstein, J. D. Campbell, and T. C. Mowry, "Programmable matter," *Computer,* vol. 38, pp. 99-101, 2005.

[48] S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji, "M-TRAN: Self-reconfigurable modular robotic system," *Mechatronics, IEEE/ASME Transactions on,* vol. 7, pp. 431-441, 2002.

[49] E. H. Østergaard, K. Kassow, R. Beck, and H. H. Lund, "Design of the ATRON lattice-based self-reconfigurable robot," *Autonomous Robots,* vol. 21, pp. 165-183, 2006.

[50] M. Yim, Y. Zhang, and D. Duff, "Modular robots," *Spectrum, IEEE,* vol. 39, pp. 30-34, 2002.

[51] J. Suthakorn, Y. T. Kwon, and G. S. Chirikjian, "A semi-autonomous replicating robotic system," in *Computational Intelligence in Robotics and Automation, 2003. Proceedings. 2003 IEEE International Symposium on,* 2003, pp. 776-781.

[52] K. Lee and G. S. Chirikjian, "An autonomous robot that duplicates itself from low-complexity components," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on,* 2010, pp. 2771-2776.

[53] A. Liu, M. Sterling, D. Kim, A. Pierpont, A. Schlothauer, M. Moses, *et al.*, "A memoryless robot that assembles seven subsystems to copy itself," in *Assembly and Manufacturing, 2007. ISAM'07. IEEE International Symposium on*, 2007, pp. 264-269.

[54] V. M. Trifa, C. M. Cianci, and D. Guinard, "Dynamic control of a robotic swarm using a service-oriented architecture," in *13th International Symposium on Artificial Life and Robotics (AROB 2008)*, 2008.

[55] Z. Kulis, V. Manikonda, B. Azimi-Sadjadi, and P. Ranjan, "The distributed control framework: a software infrastructure for agent-based distributed control and robotics," in *American Control Conference, 2008*, 2008, pp. 1329-1336.

[56] N. Lenzi, B. Bachrach, and V. Manikonda, "DCF (Registered)-A JAUS and TENA Compliant Agent-Based Framework for Test and Evaluation of Unmanned Vehicles," DTIC Document2011.

[57] P. Nebot and E. Cervera, "Agent-based application framework for multiple mobile robots cooperation," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, 2005, pp. 1509-1514.

[58] X. L. Tao Zhang, Yi Zhu, Xiaqin Li, Song Chen, "Coordinative Control for Multi-Robot System through Network Software Platform," *iConcept Press*, pp. 51-59, 2010.

[59] M. INC, "Documentation & Technical Support for MobileRobots Research Platforms," ed: Adept MobileRobots INC 2006.

[60] J.-C. Baillie, "The URBI Tutorial," ed: Gostai, 2006.

[61] B. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *Proceedings of the 11th international conference on advanced robotics*, 2003, pp. 317-323.

[62] D. Blank, D. Kumar, L. Meeden, and H. Yanco, "Pyro: A python-based versatile programming environment for teaching robotics," *Journal on Educational Resources in Computing (JERIC)*, vol. 4, p. 3, 2004.

[63] A. Elkady, J. Joy, and T. Sobh, "A plug and play middleware for sensory modules, actuation platforms and task descriptions in robotic manipulation platforms," in *Submitted to Proc. 2011 ASME International Design Engineering Technical Conf. and Computers and Information in Engineering Conf.(IDETC/CIE'11)*, 2011.

[64] B. Chen, H. H. Cheng, and J. Palen, "Mobile-C: a mobile agent platform for mobile C/C++ agents," *Software: Practice and Experience*, vol. 36, pp. 1711-1733, 2006.

[65] J. Von Neumann and O. Morgenstern, "Theory of games and economic behavior," *Bull. Amer. Math. Soc*, vol. 51, pp. 498-504, 1945.

[66] K. Kuwabara, T. Ishida, and N. Osato, "AgenTalk: Coordination Protocol Description for Multiagent Systems," in *ICMAS*, 1995, pp. 455-461.

[67] J. Juneja. Available: http://www.jaijuneja.com/blog/2013/05/simultaneous-localisation-mapping-matlab/

[68] L. E. Parker, "ALLIANCE: An architecture for fault tolerant multirobot cooperation," *Robotics and Automation, IEEE Transactions on*, vol. 14, pp. 220-240, 1998.

[69] L. Jiang and R. Zhan, "An autonomous task allocation for multi-robot system," *Journal of Computational Information Systems*, vol. 7, pp. 3747-3753, 2011.