ASEE 2014 Zone I Conference, April 3-5, 2014, University of Bridgeport, Bridgpeort, CT, USA.

UBSwarm: Design of a Software Environment to Deploy Multiple Decentralized Robots

Tamer Abukhalil
Robotics, Intelligent Sensing & Control (RISC) Laboratory
School of Engineering, University of Bridgeport
221 University Avenue, Bridgeport, USA
tabukhal@my.bridgeport.edu

Madhav Patil
Robotics, Intelligent Sensing & Control (RISC) Laboratory
School of Engineering, University of Bridgeport
221 University Avenue, Bridgeport, USA
mpatil@my.bridgeport.edu

***Abstract*— this article presents a high-level configuration and task assignment software package that distributes algorithms on a swarm of robots which allows them to operate in a swarm fashion. When the swarm robotic system adopts a decentralized approach, the desired collective behaviors emerge from local decisions made by the robots themselves according to their environment. This paper first brings a discussion on the existing swarm control environments. Secondly it proposes a software application that aims to facilitate the deployment of multiple robotic agents which have different configurations and sensory components. Using its GUI, the proposed system expects the operator to select between several available robot agents and assign the group of robots a particular task from a set of available ones. 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.**

Keywords— Decentralized Swarm Intelligence, Modular Robotic Agents, Robotics Interactive Software, Robots Deployment Environment.

I. INTRODUCTION

It has been proven that a single robot with multiple capabilities cannot necessarily complete an intended job using the same time and cost as that of multiple robotic agents. Different robots, each one with its own configuration, are more flexible, robust and cost-effective. Moreover, the desired tasks may be too complex for one single robot, whereas it can be effectively done by multiple robots [1, 2]. One of the key advantages of the corporative multi-agents robotic systems is fault-tolerance in which a robot can take over the task of a failing one. Modular robotic systems have shown to be robust and flexible in the tasks of localization, surveillance [3], and reconnaissance [4]. Such properties are likely to become increasingly important in real-world robotics applications.

Decentralization means that the algorithm does not require access to the full global state and all control computations are done locally. However, 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 paper 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. Our objective is to develop algorithms that can provide connectivity between multiple agents, besides building central software to track these agents. Such system design is motivated by our interest in multi-robot control for the deployment of potentially large numbers of cooperating robots and application tasks such as persistent navigation, object manipulation, and transportation. Online algorithms operate under the assumption that future events (inputs) are uncertain. Hence, they will occasionally perform an expensive operation to efficiently respond to future operation. Generic and parameterized algorithms provide behaviors that are parameterized.

In the following section we provide a short analysis of existing swarm deployment environments. In section III we present a deployment software package for obtaining decentralized control that can provide interesting collective behaviors dedicated to different tasks/applications with a new collective and mobile reconfigurable robotic system. We do not consider any particular hardware or infrastructure of each swarm agent, as our focus is building control mechanisms that allow the system to operate several simple heterogeneous agents. In section IV we evaluate *UBSwarm* framework with respect to human rescue and wall painting applications. Finally, Section V presents a summary of the work and draws some conclusions.

II. RELATED WORK

A comprehensive investigation and evaluation of the present multi-robotic systems (MRS) has been thoroughly discussed in our previous work [5]. In that survey we organized and classified ten swarm robotics systems and their corresponding behavioral algorithms into a preliminary taxonomy. We concluded that several algorithms have been developed to run on swarms of robots. These algorithms varied in complexity. Some provided basic functionality, such as leader following, while others exhibited complex interactions between the team of robots such as bidding on tasks according to arbitrary rules. Many early approaches in the literature concentrated on behavior-based technique where several desired behaviors are prescribed for each agent, and the final control is derived from a weighting of the relative importance of each behavior. On the other hand, recent researchers have begun to take a system controls perspective and analyze the stability of multiple robot agents. Other important hardware aspects of the current modular swarm robotic systems such as self-reconfigurability, self-replication, self-assembly, cost and miniaturization with robustness, flexibility, and scalability were thoroughly analyzed in our other work [6].

Some script-based robot programming was designed specifically for robotic control like Pyro[7]. 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. [8], have developed a framework to utilize and configure 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 author's 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 [9] where the system is capable of interacting with multiple robots using Mobile-C library [10], 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.

Gregory P. Ball G. et al. [11], have proposed an 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. 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 Logic System to parse input into valid commands, and a Robot Server, which receives commands from the Logic System and communicates these commands to the appropriate robot. Local components communicate using direct procedure calls.

III. METHODOLOGY

We are developing an environment to utilize robots that have different modular design and configuration of sensory modules, and actuators. The system will be implemented as a GUI interface to reduce efforts in controlling swarm robotic systems. The proposed application offers customization for robotic platforms by simply defining the available sensing devices, actuation devices, and 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 platform, operating systems, or the introduction of new functionalities. The basic hierarchy of the UBSwarm deployment platform is shown in Fig. 1.

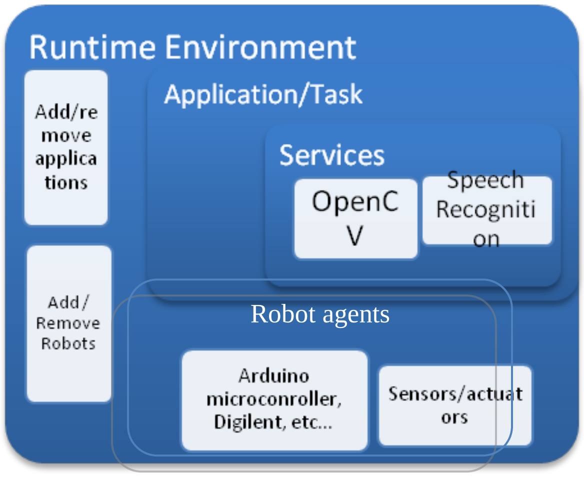


Fig. 1: High-End System Overview

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 systems and protocols. 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 or communicates/monitors the robots through the USB (serial port), RF, WiFi, or Bluetooth. In our experiment we used our own robot agents that incorporate Arduino and Digilent Max32 microcontrollers.

UBSwarm provide a direct forward two-step configuration that helps the operator to select between several available robot computers (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.

IV. System Architecture

UBSwarm is an interactive Java-based application designed for extensibility and platform independence. The system establishes communications with embedded robot modules via various mediums. At the time of startup the system will expect the operator to:

- Configure the system by picking the available agents, their onboard features (sensors, motors, etc.) and the services needed to accomplish each task
- Or simply run the system using the last executed configuration.

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 that the platforms will 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 a component interface for the upper layers call.

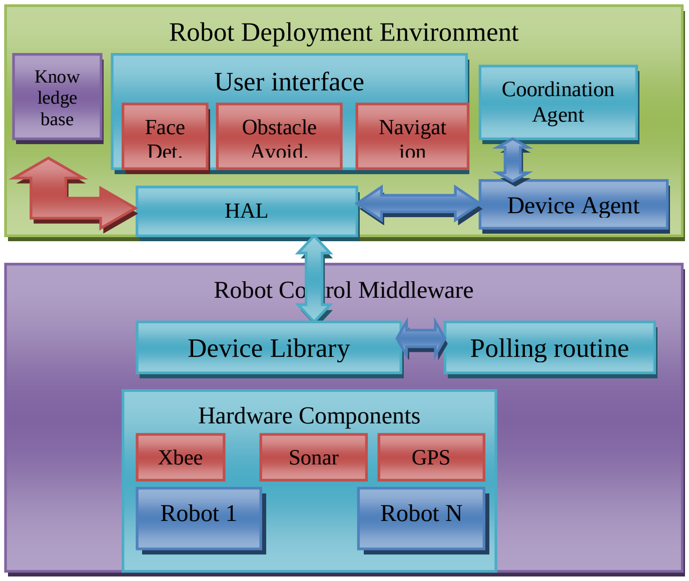


Fig. 2: System Architecture

A. *Robot Deployment System*

The deployment system interacts with agents through various types of communication mediums. The deployment system takes the responsibility of running actions according to the definition parameters and the different 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 algorithm; 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 Fig.2. The deployment system contains the developer interface, the coordination agent, the dynamic interpreter, and the knowledge base.

1) *Operator Interface*

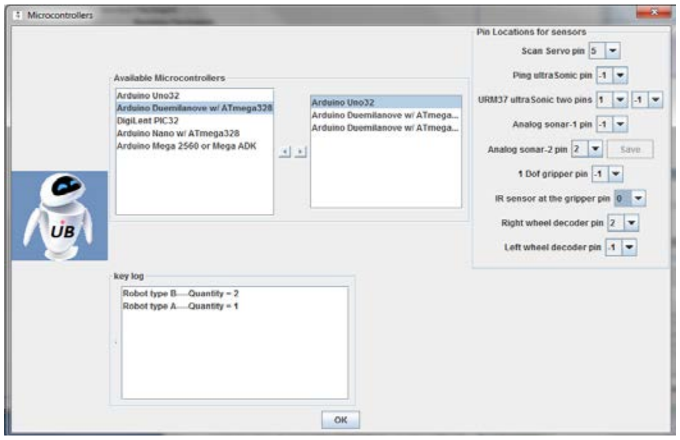
The system developer interface provides the human operator command and control windows. The user can interact with the computer through interaction tools which provides a list of actions/tasks and the available robotic agents. In some other parts of the interface, the user will be prompted to input the 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). 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 by manipulating the arrow buttons as shown in fig. 3 (a). 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 computer will ask the user to connect each robot to allow uploading the program as shown in fig. 3 (b). 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.

2) Coordination Agent

The heterogeneity of the robots and the operating platforms imposes dependencies such as data format, location of machine addresses, and availability of the components. As addressed in [8] the data format dependency is removed by a standard data format that is machine independent. Just like the functional modules described earlier, the data format is regarded as a component in the system. 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. In [8] the other two dependencies are removed by having all information being sent to an addressable channel instead of sending information directly to a specific robot. 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.

3) Runtime Interpreter

When new devices are plugged in, system developers can install new platform software packages specific for the execution of the newly added devices. In other words, 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. 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.



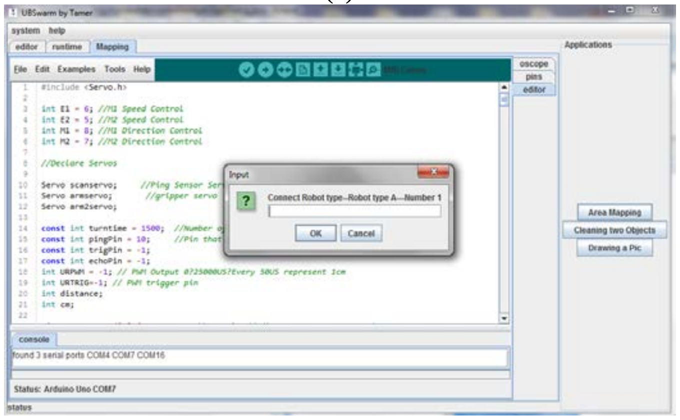
Arduino Uno32
Arduino Duemilanove w/ ATmega328
DigiLent PIC32
Arduino Nano w/ ATmega328
Arduino Mega 2560 or Mega ADK

Arduino Uno32
Arduino Duemilanove w/ ATmega...
Arduino Duemilanove w/ ATmega...

Scan Servo pin	5
Ping ultraSonic pin	-1
URM37 ultraSonic two pins	1, -1
Analog sonar-1 pin	-1
Analog sonar-2 pin	2
1 Dof gripper pin	-1
IR sensor at the gripper pin	0
Right wheel decoder pin	2
Left wheel decoder pin	-1

Robot type B—Quantity = 2
Robot type A—Quantity = 1

(a)



```
#include <Servo.h>

int E1 = 6; //M1 Speed Control
int E2 = 5; //M2 Speed Control
int M1 = 8; //M1 Direction Control
int M2 = 7; //M2 Direction Control

//Declare Servos

Servo scanservo;     //Ping Sensor Ser
Servo armservo;       //gripper servo
Servo arm2servo;

const int turntime = 1500;  //Number o
const int pingPin = 10;     //Pin that
const int trigPin = -1;
const int echoPin = -1;
int URPWM = -1; // PWM Output 0?25000US?Every 50US represent 1cm
int URTRIG=-1; // PWM trigger pin
int distance;
int cm;
```

Connect Robot type--Robot type A—Number 1

found 3 serial ports COM4 COM7 COM16

Status: Arduino Uno COM7

status

(b)

Fig. 3: (a) Operator Interface configuration (b) Prompt to connect and upload programs

The flow of information managed by the dynamic interpreter is shown in fig. 4. 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 an- other application of a change in this application and the *Request-Reply message*, used when an application sends back a reply. The messages are classified into three categories: Simple message (short messages with low delay requirements), real-time message (short 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. Fig. 4 shows the operation of the runtime interpreter when services are added to the system.

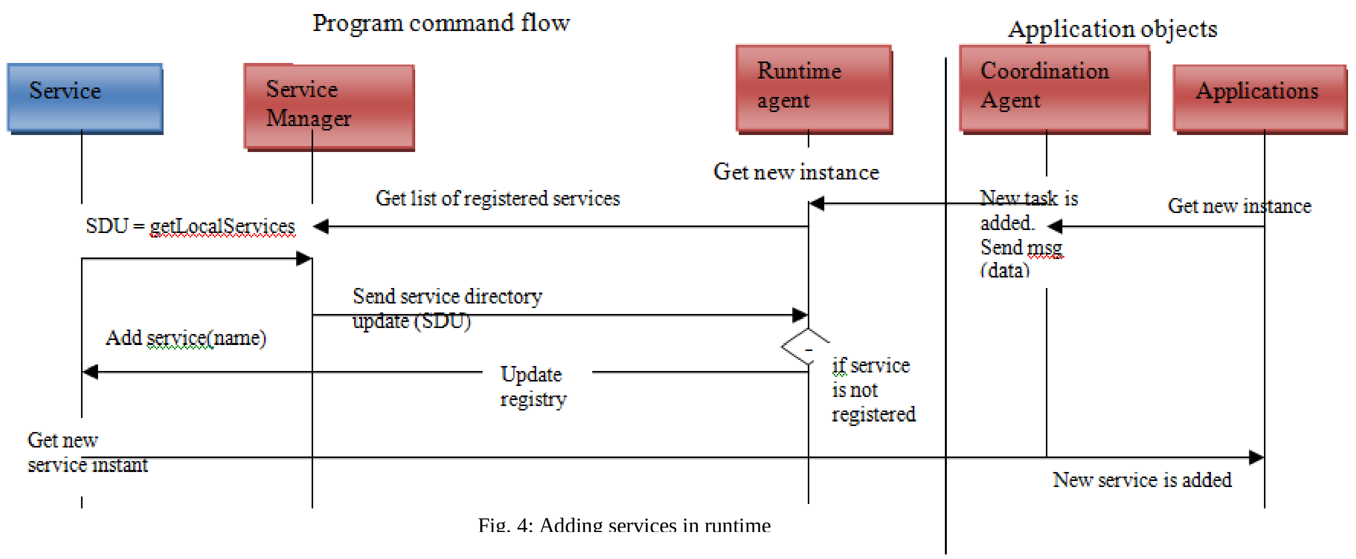


Fig. 4: Adding services in runtime

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 [8] each component has an XML configuration file to customize its behavior. Each component is designed to be dynamically reconfigurable by the dynamic interpreter during robot operation.

4) *Knowledge Base (Registry)*

The Knowledge base contains all of the necessary information for each robot to give the operator the ability to address each task. This includes a listing of all possible actions, service modules, and behavioral components implementations for each robot. The knowledge base 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.

V. THE EXPERIMENT

A. *Human Rescue (transporting a human)*

The task of human/object transporting requires the robotic agents to cooperatively work together pull the heavy object in order to reach a desired position. As shown in fig. 4 the swarm of robots generates simple actions based on observations from its environment. The algorithm was developed for a group of robots to autonomously cooperate such that the pulled object can be positioned and oriented in the 2D space. Corporation between robots is achieved by exchanging messages when additional robots are needed to pull the object. Each individual robot is programmed to call another one if its wheel/tread on one side rotates in higher speed than the other side. The experiment begins when the robot start looking for a human as it wanders in the unknown environment; the robot is equipped with onboard camera in order to detect the human body on the ground. The robot will then call another agent using Xbee-based communication and then the two robots will place themselves on different corners and using their grippers and by sending a particular synchronization message, the two will attach to the body and start pulling backward to a safer position. A human prototype was built and several experiments were conducted. More weights have been placed inside the human prototype as the number of the robots increased. We noticed that the configuration that uses more than two robots is able to pulling heavier objects however; this configuration causes the robot to skid to any of the both sides and consequently this act reduces the elapsed distance the robots have actually pulled. Table 5.1 below shows the distances achieved by the different number of robots with respect to different weights for the object being transported.

Table 1: Successful pulling distance according to different number of robotic agents

Object weight	***Pulling Distance***				
	1 robot	***2 robots***	***3 robots***	***4 robots***	***5 robots***
10 Kilograms	***0.1 Meters***	***1.5***	***4***	***3***	***1***
20 Kilograms	***0***	***0.5***	***4***	***2***	***1***
40 Kilograms	***0***	***0***	***2***	***2***	***1***

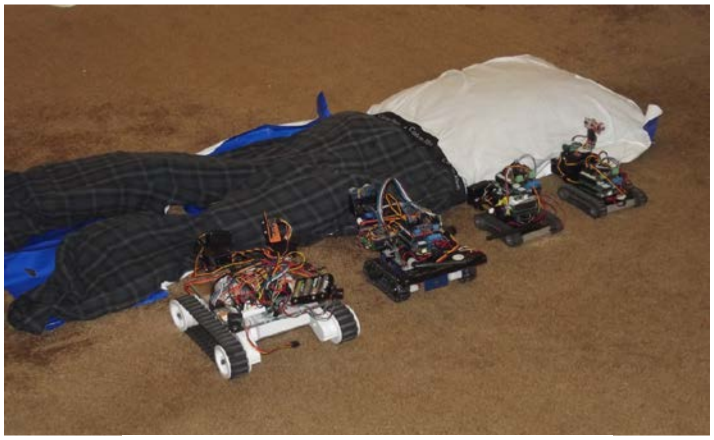


Fig. 4: Human rescue using 4 robots

B. *Wall Painting*

The task is executing interior painting tasks by our robotic swarm system. As we mentioned earlier our robotic agents each have location sensors, simple communication modules, and vision capability to be able to move away from each and start painting their little part of the wall in parallel. Experiments were conducted in order to examine the integration between humans and the robot in their work. Painting using a brush is the most commonly used by human workers. Using a brush in our system requires more sophisticated robotic arms. Painting by a spraying, however, is less demanding in terms of accuracy, and therefore more appropriate for our system.

1. *Arm and End effecter*

The end effecter is the basic 1-Dof gripper attached to a 2-Dof arm that controls the position of the end effecter in two movements; up, down, and 360 degree rotation of the gripper around its own center. Fig. 5 shows the movements and the offsets along direct Z axis.

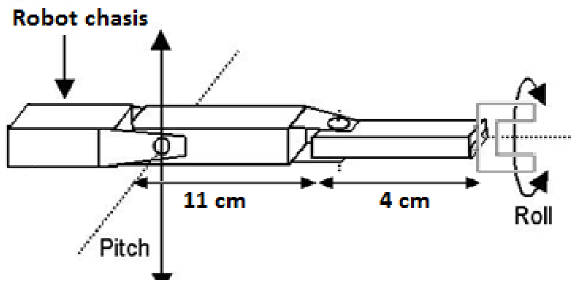


Fig. 5: The 2-Dof sketch for the robot arm

Painting with that particular type of end-effecter creates multiple adjacent rectangular coating sectors as shown in fig. 6. 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.

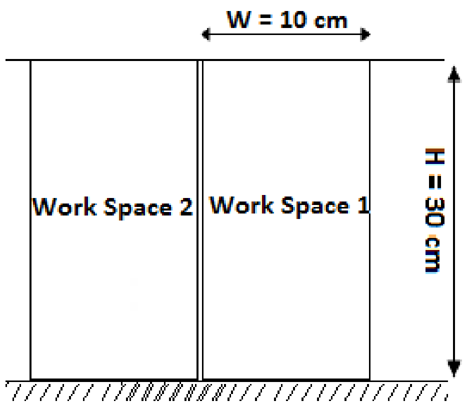


Fig. 6: The surface covered by the painter

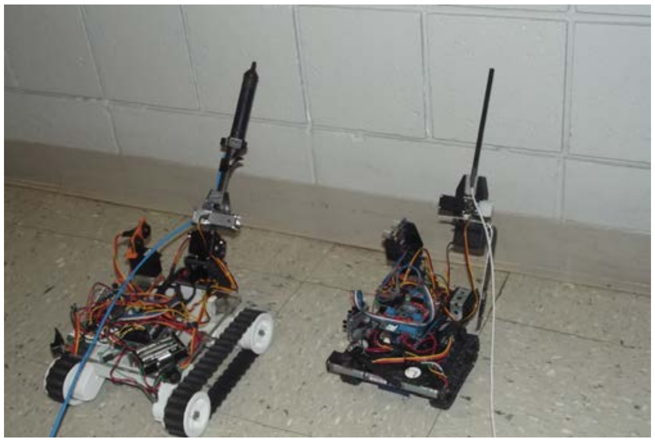


Fig. 7: the Nozzles attached to the robots

Fig. 7 shows two robots performing a painting test using the nozzles attached to each of their grippers.

2. *Painting method*

Two robots were used each equipped with the arm discussed earlier and a flexible hose attached to the end effecter and one end and to a compressed paint container at 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:

- Each robot moves concurrently at the same speed with the other robots. An infra-red 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.
- 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 movement of the arm and the distance from the wall.

- 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 were interested in learning how much time is saved when painting using multiple robots. To do so, we ran two experiments. The first one which involved one single robot, the task was completed in 30 minutes. In the second experiment we ran two robots in which they took 14 minutes to complete their task.

VI. Conclusion

We learned, as experimental results depict, the sensing and the overall task-specific capabilities of the platforms can be easily upgraded by adding another sensor, *e.g.*, laser range finders and that is how the name extendible robots come from. UBSwarm environment generates programs that cope with changes of the robots configurations. Moreover, better design considerations has to take place when building a robot that is intended to accomplish tasks that incorporate another robotic agents as a group. Using multi robotic systems in assigned missions may cause some actual collisions. This problem was illustrated in the human-rescue task as the robots tended to slip and skid to one side as they pulled the heavy object. In future work we will attempt to design a mechanism that gets activated should the robot start slipping.

References

[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] 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.

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

[5] T. Abukhalil, M. Patil, and T. Sobh, "Survey on Decentralized Modular Swarm Robots and Control Interfaces," *International Journal of Engineering (IJE)*, vol. 7, p. 44, 2013.

[6] M. Patil, T. Abukhalil, and T. Sobh, "Hardware Architecture Review of Swarm Robotics System: Self-Reconfigurability, Self-Reassembly, and Self-Replication," *ISRN Robotics*, vol. 2013, 2013.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.