Software System Integration of Heterogeneous Swarms of Robots

Mihai Gânsari, Cătălin Buiu

Department of Automatic Control and Systems Engineering, Politehnica University of Bucharest, Romania, (e-mails: (mihai.gansari, catalin.buiu)@acse.pub.ro).

Abstract: Swarm robotics has gained momentum in the last years due to impressive technological and scientific advances. A wide range of application areas benefit from the power of collective behaviors and emergent intelligence even if individual agents in a swarm are simple and have limited sensing and processing capabilities. This paper presents a new approach to the problem of software system integration of heterogeneous robots in a swarm. We showcase the capabilities of this approach by providing a sound methodology, full code and demonstration videos of the run-time platform through experiments with heterogeneous swarms. The proposed solution is shown to be modular, scalable, robust, and flexible.

Keywords: mobile robots, swarm robotics, software integration, robot operating system.

1. INTRODUCTION

Swarm robotics can be defined as "the study of how a large number of relatively simple physically embodied agents can be designed such that a desired collective behavior emerges from the local interactions among agents and between the agents and the environment" (Şahin and Spears, 2005) without any centralized control or use of global information. The emergent behaviors of such swarms demonstrate robustness and flexibility which are desired qualities in real-world applications, such as warehouse automation or civilian or military search and rescue missions.

To date, a lot of research effort has been spent on homogeneous robot swarms (where all the robots are identical) and it is considered that in order to reach the complexity of real-world systems it is necessary to address the stringent issue of designing integrated heterogeneous robot swarms in which robots can differ physically, functionally and informationally (Dorigo et al., 2013). Integration in this context relates to the fact that each individual robot or subgroup of similar robots can communicate and interact with other robots and the swarm as a whole, regardless of their hardware and software characteristics. Fig. 1 shows a heterogeneous swarm in which there are three different types of robots (Koala, Khepera, and E-puck) which differ in terms of size, sensors, processing power, capabilities and autonomy. Even for identical robots, they can differ in terms of processing power, e.g. the two Khepera robots in Fig. 1, one of which is a Raspberry Pi-powered robot.

A list of issues and challenges in designing heterogeneous robotic swarms, which include the physical and behavioural integration among various hardware platforms, with different components, capabilities, software interfaces and multiple, possibly competing, requirements, is detailed in (Dorigo et al., 2013). Given the difficulty of hardware and software integration, it is not surprising that only a few papers have approached the problem of designing and deploying heterogeneous robot swarms, from both the hardware and software points of view. In (Patil and Abukhalil, 2014), the hardware architecture of a swarm of five different mobile robots is addressed, while this issue is approached from a system-of-systems perspective in (Sahin et al., 2007). Sharing of information by multiple robots is addressed using a graph-based map-building method in (Moon et al., 2015).

Fig. 1. A heterogeneous robotic swarm.

The software issue of deploying heterogeneous robot swarms is addressed through a programmability perspective in (Pinciroli et al., 2015), where an extensible programming language for self-organizing heterogeneous robot swarms is presented as an indispensable tool. A prominent actor on the robotics market is ROS, an open-source Robot Operating System which allows compatible software to be written for different robots by providing libraries and tools to assist developers to create robot applications and reducing the development time (Quigley et al., 2009), (Mayachita et al., 2013), (Alisher et al., 2015), (Araújo et al., 2014), (Tsarouchi et al., 2015). A real-time extension of ROS is presented in (Wei et al., 2016).

The main contribution of this paper is to propose and demonstrate the viability of a software framework which allows the integration of various robots into a unitary swarm of robots.

The paper is structured as follows. The next section gives an overview of the currently available robot hardware and software tools used to integrate software components in a heterogeneous robot swarm. The third section presents the software implementation for the robots in order to make them ROS compatible. The fourth and fifth sections present functioning modes for the heterogeneous swarm in multiple configurations and the test results. The paper ends with conclusions and directions for further improvements.

2. OVERALL SYSTEM ARCHITECTURE

This section defines and describes a possible view of a heterogeneous robot swarm in terms of structure, components, their properties, and the relationships between them. The emphasis will be placed on programming the interconnection of various robots in a common software framework.

2.1 System Overview

The goal is to have different robots interacting in an integrated way which can enable the effective realization of high-level, complex goals in a dynamic environment. Some hardware additions might be needed but not necessarily in all cases, as the majority of robots are built around connectivity, hardware wise, using standard devices, as will be explained in the next subsections.

Regarding software, a common framework is required. This is accomplished using ROS, a very powerful tool in robotics software development which has become nearly a standard in robotics research.

2.2 Robot Operating System

ROS (http://wiki.ros.org/) is an open-source robotics software framework for robotics software development. It provides the user with connectivity and control of robot devices through a common software interface. ROS offers message passing (publish/subscribe) between processes (nodes), software libraries and tools for writing, building and running programs. It also offers visualization, data recording and other tools for analysing the results and for adjusting the code. Software package management (catkin, rosbuild) is another important feature. ROS also offers network connectivity between machines, allowing the software to be run across multiple machines.

ROS is an open-source and research community driven software, and is written in the very popular and powerful C++ and Python programming languages, but it is not an operating system in itself, as it runs on the Linux operating system, especially on the Ubuntu distribution. It is called so because it offers many operating system-like services between the computer/robot and the user.

The power of ROS is that it allows the user to freely use all the existing ROS packages, modify or create new ones. Linking the functionalities of these packages makes ROS a great tool for rapid robotics development. ROS offers the supported robots (with ROS drivers) important high-level robot application packages such as robot vision, navigation and planning, intelligent control, grasping, and SLAM (Simultaneous Localization and Mapping). It also offers links to other existing robotics frameworks and simulators such as Player, Gazebo, Webots etc.

In order to have a robot using the ROS software, a software–hardware link (driver) between the computer running ROS and the robot is needed. The robot devices that need to be used or controlled (e.g. camera, motors) must be connected directly to this computer via USB, serial, or other ports, or have the robot's central (embedded) computer connected to this computer. The embedded computer controlling all the robot devices is more common because robots are usually built to be standalone machines. The ROS driver implementation for the robots presented in this paper will be described in the next section.

Another advantage of ROS is that it is made to scale easily and appropriately for various size systems. Robots usually have embedded devices and software, and this provides real-time features. ROS is intended for higher-level application use, it is a fast software but not real-time. This might be a limitation, and some ROS embedded tools might be needed for this.

2.3 Test Robots

The majority of robots used by the industry and research are built and mass-produced by robotics companies. These robots come with built-in software for their microcomputer architecture to connect all the on-board sensors and actuators, and even provide purchasable extensions, such as more sensors, e.g. laser scanners. The application purposes and research areas in which these robots are used lead in many cases to the robots needing modifications or add-ons to their hardware and software, whether they are open-source platforms or not. Depending on the robot architecture, the firmware may be easily modified, with software tools and programming hardware either built-in or external. Modifying the robot firmware is good for program running speed and real-time restrictions. When working on advanced, high-level applications, where developers and users need to visualize what the robot is doing, modifying the firmware is not a good solution. With older hardware, the memory of the embedded computer is usually limited. High-level programming tools, such as C++, Python programming languages and other object-oriented programming languages are recommended in these cases. For these tools a powerful computer is needed and if it is not integrated in the robot, this computer needs to be added to it.

The robots used in this paper to demonstrate and validate the proposed software system integration are three types of mobile robots of different sizes and capabilities and are shown in Fig. 2.

The smallest robot used is the E-puck robot (e-puck.org), which is an open-source robot. It has a dsPIC microcontroller connected to 2 stepper motors and the following sensors: IRs, VGA camera, microphones. It also has a speaker and LEDs. The robot supports serial communication and Bluetooth. The

firmware for this robot is written in C programming language. Programs can be written and cross-compiled to create a .hex file to write to the flash memory of the microcontroller via Bluetooth, serial or using a programmer device wired directly to the robot.

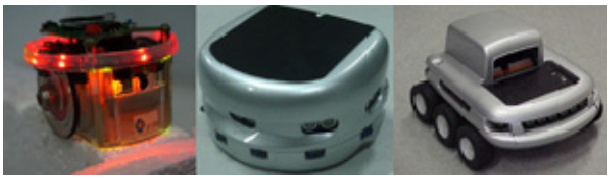

Fig. 2. (Left to right) E-puck, Khepera III and Koala robots.

The second robot is a larger but still small size robot Khepera III (K3). The K3 is a proprietary developed research robot and it has a main dsPIC microcontroller connected to two other PIC microcontrollers for each of its 2 motors with encoders, and the following sensors: IR, ultrasonic. The robot supports serial, Bluetooth and USB communication. The robot has an installed firmware for all the robot capabilities and can be updated. A Korebot board which has an ARM processor can be attached to the robot to run programs and control the robot. The board runs an Ångström Linux distribution OS. C programs can be written using the Korebot API (K-Team, 2012).

The last robot is the Koala, which is another proprietary research robot. The Koala is a medium size robot. The robot has a Motorola microcontroller connected to 2 motors with encoders, and the following sensors: IR, temperature, torque. The robot supports serial and Bluetooth communication. The microcontroller runs a software named Koala BIOS (K-Team SA, 2001) written in Assembly language or C. This BIOS can be modified and used for other programs written in C and compiled with a GNU C cross-compiler.

The three robots support receiving commands via serial/Bluetooth interfaces. These commands can fully use all the on-board capabilities of the robots and can be used in any software program that uses serial communication or has software libraries for it.

2.4 Robot Hardware Modifications for Software Requirements

All the robots have IR proximity sensors and serial over Bluetooth communication for PC connectivity. In order for the robots to communicate, additional software and/or hardware is required.

Bluetooth and Wi-Fi device software drivers are implemented already in major operating systems (Linux, Windows, Mac OS). ROS is also a common software interface for robots which can be used to communicate using the wireless devices. Linux is preferred because it is open-source, it is supported on multiple micro-computer architectures, and ROS is implemented for the Ubuntu distribution of Linux. Thus platforms running Ubuntu are required.

The smaller robots, E-puck and K3, can run Linux on an attached small size Raspberry Pi computer board (version 2 B+ 2015 is recommended for more computing power) with USB Bluetooth and Wi-Fi dongles attached. The K3 already runs a version of Linux on its Korebot computer board, but it lacks computing power and a very small version of ROS would need to be compiled for its processor, thus a Raspberry Pi is preferred.

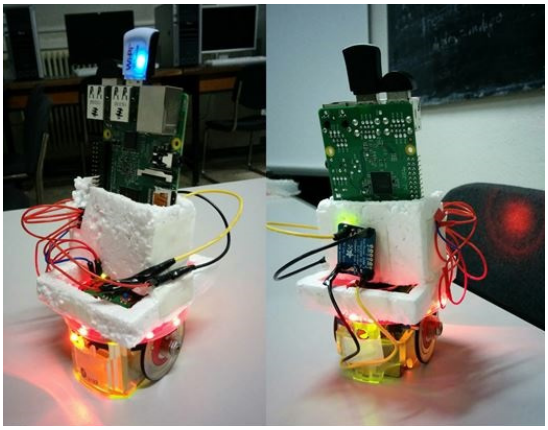

Fig. 3. Modified E-puck with Raspberry Pi 2 B and Wi-Fi dongle (front and back).

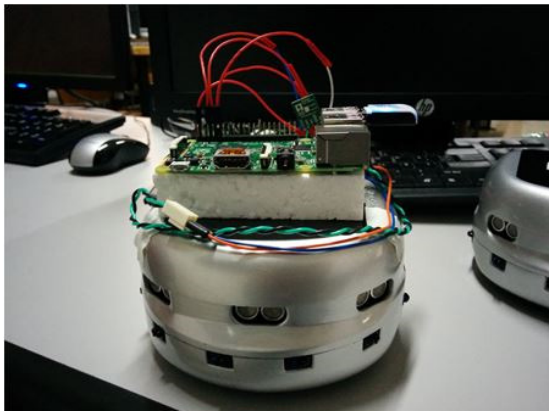

Fig. 4. Modified Khepera III robot with Raspberry Pi 2 B and Wi-Fi dongle.

Fig. 5. Koala Rzobot connected via serial to a 13.3” laptop mounted on a support. A Kinect sensor is mounted on the robot and connected directly to the laptop via USB.

The Koala robot, being larger in size, can carry a small size laptop on it. A support was added to the robot to carry the laptop. Having a laptop PC adds a lot of power, versatility and connectivity to the robot. Such an example is adding a

Kinect sensor for precise mapping information, needed for SLAM.

Pictures of the modified standard robots are shown in Fig. 3–5, and Table 1 summarizes all their characteristics.

Table 1. Overview of robots used.

	E-puck	Khepera III (K3)	Koala
Embedded Computer	dsPIC30F6014A at 60 MHz, 8KB RAM, 144KB flash memory	dsPIC30F5011, 60 MHz, 4KB RAM, 66KB flash memory	M68331 32-bit MCU at 22MHz, 1MB RAM, 1MB flash memory
Connectivity	Bluetooth, Serial	Bluetooth, Serial	Serial
Extensions	-	Korebot: Xscale PXA255 400 MHz, 64MB RAM, 32 MB flash, Wi-Fi card	connectBlue industrial Bluetooth 333s
Robot Additions			
Computer	Raspberry Pi 2 B 900MHz quad-core ARM Cortex-A7 CPU, 1GB RAM, 32GB SD-card, USB Wi-Fi dongle, Ubuntu-ARM 14.04 OS	Raspberry Pi 2 B 900MHz quad-core ARM Cortex-A7 CPU, 1GB RAM, 32GB SD-card, USB Wi-Fi dongle, Ubuntu-ARM 14.04 OS	Laptop Z30t-B-104, 13.3" monitor, Intel i7-5500U, 16GB RAM, 512 SSD, Wi-Fi, Bluetooth, USB 3.0, Ubuntu 64-bit 14.04 OS
Sensors	-	-	Kinect

Any number of the robots from the above table can be used with any additional robots that are ROS compatible.

3. ROS LINK. SOFTWARE IMPLEMENTATION

The robots presented in Section 2 must be able to communicate with each other. For this they need ROS drivers to link them to the software environment and to other robots in the swarm. These ROS drivers are sometimes made by the companies that produce the robots or research groups that use the robots, and this is the case of the E-puck robot.

Starting in 2015, ROS drivers for the E-puck robot are available. These drivers are written in Python (https://github.com/verlab-ros-pkg/epuck_driver) and C++ (https://github.com/gctronic/epuck_driver_cpp). They use the Bluetooth communication and serial over Bluetooth commands to control and receive information from the sensors.

The K3 and Koala robots also need software drivers for the ROS link. The ROS drivers are written in C++ and use serial commands (Fig. 6) to control the robots and read sensor information. A serial communication library from ROS is used (http://wiki.ros.org/serial). The code is similar for the robots, with only some commands and functionalities differing because the robots have different on-board devices, actuators and sensors, needing different values for their data, thus different drivers. The code is a basic C++ program which controls the robot using a serial library, but the addition is that it links the functions and data to ROS functions in the same program.

D Set speed
Format of the command: D,speed_motor_left,speed_motor_right CR/LF
Format of the response: d CR+LF
Effect: Set the speed of the two motors. The unit is the pulse/10 ms that corresponds to 4.5 millimetres per second.

E Read speed
Format of the command: E CR/LF
Format of the response: e, speed_motor_left, speed_motor_right CR+LF
Effect: Read the instantaneous speed of the two motors. The unit is the pulse/10 ms that corresponds to 4.5 millimetres per second.

N Read proximity sensors
Format of the command: N CR/LF
Format of the response:
n,val_sens_L0,val_sens_L1,val_sens_L2,val_sens_L3,val_sens_L4,
val_sens_L5,val_sens_L6,val_sens_L7,val_sens_R0,val_sens_R1,val_sens_
R2,val_sens_R3,val_sens_R4,val_sens_R5,val_sens_R6,val_sens_R7
CR+LF
Effect: Read the 10 bit values of the 16 proximity sensors (section 2.1.6), from the front left sensor to the back left sensor, then from the front right sensor to the back right sensor.

Fig. 6. Koala motors and sensors commands (K-Team SA, 2001), pp. 39 and 41.

The source program and other configuration files are placed in a node folder and structured for the ROS catkin workspace (http://wiki.ros.org/catkin) in order for the project node to be built and linked to ROS.

The driver node can also contain messages files to pass between ROS programs and roslaunch files to easily and automatically launch every needed node and program that uses this driver.

A basic Koala robot ROS driver C++ code is listed in the lines shown in Fig 7 and 8.

The program subscribes to and waits for general ROS geometry_msgs/Twist.msg which it uses as commands for the left and right motors. The Koala robot is connected to the mounted computer using a robot serial cable and a serial-to-USB cable, thus the /dev/ttyUSB0 port is used in the driver.

Having the robot driver node connected to ROS, other programs/applications can listen to this node, which can publish information on ROS topics, and wait for motor commands over the motor topic.

```
#include <time.h>
#include <string.h>
#include <iostream>
#include <stdio.h>
#include <serial/serial.h>
#include <geometry_msgs/Twist.h>
#include <ros/ros.h>

#define WHEEL_DIAMETER 8.5       // cm.
#define WHEEL_SEPARATION 29    // Separation
between wheels (cm).

using namespace serial;
using namespace std;
using namespace ros;

int speedLeft = 0, speedRight = 0;

// port, baudrate, timeout in milliseconds
Serial koala_ser("/dev/ttyUSB0",115200,
serial::Timeout::simpleTimeout(1000));

ros::Subscriber cmdvelSubscriber;

//function to write serial commands for motors
void koalaMotors(){
      char sercmd[14];
      sprintf(sercmd,"D,%d,%d\n",speedLeft,speedRight);
      koala_ser.write(sercmd);
      }

//function to get left and right motor speeds from ROS twist
msgs
void handlerVelocity(const
geometry_msgs::Twist::ConstPtr& msg)
{// Controls the velocity of each wheel based on linear and
angular velocities.
   double linear = msg->linear.x;
   double angular = msg->angular.z/4; //the values are
adjusted by testing
   cout<<"linear="<<linear<<" angular="<<angular<<endl;

   // Differential robot kinematic model
   double leftwheel = (linear - ((WHEEL_SEPARATION /
2.0) * angular)) /(WHEEL_DIAMETER);
   double rightwheel = (linear + ((WHEEL_SEPARATION
/ 2.0) * angular)) /(WHEEL_DIAMETER);

  if(int(leftwheel*200)>100){speedLeft=100;}
   else if (int(leftwheel*200)<-100){speedLeft=-100;}
   else {speedLeft = int(leftwheel*200);}

   if(int(rightwheel*200)>100){speedRight=100;}
   else if (int(rightwheel*200)<-100){speedRight=-100;}
   else {speedRight = int(rightwheel*200);}

   cout << "Koala - new speed: " << speedLeft << ", " <<
speedRight << endl;
   }
```

Fig. 7. koala.cpp (part 1).

```
int main(int argc, char **argv){
     cout<< "Koala is Connected on Port:" <<
koala_ser.getPort()  << "at Baudrate:" <<
koala_ser.getBaudrate() <<endl;
     //test serial command to the motors, spin the robot
     //koala_ser.write("D,100,-100\n");
     //ros::Duration(3).sleep();

     ros::init(argc, argv, "koala_driver");
     ros::NodeHandle n;
     cmdvelSubscriber = n.subscribe("/koala_motors", 10,
handlerVelocity);
     ros::Rate loop_rate(1000); // desired frequency for loop

     while(ros::ok()){
     koalaMotors(); //write on serial
     ros::spinOnce();//Allow ROS to check for new ROS
Messages
     loop_rate.sleep(); //Sleep for some amount of time det
by loop_rate
}
  return 0;
}
```

Fig. 8. koala.cpp (part 2).

Using this template, drivers for the K3 and E-puck were also created, and along with the Koala drivers. The drivers created with this template are available as open-source software at: https://github.com/pktehgm/newswarm/tree/master/robot_drivers.

4. USE CASE SCENARIOS

Depending on the applications and tasks required for the heterogeneous robotic swarm presented here, the robots can communicate and run programs using the ROS system in five different working modes, which we will introduce in the following. These modes are different in terms of where the programs run, on which robot and how. The programs refer to the drivers, applications and ROS. The software drivers for controlling the robots and programs for higher-level applications and tasks can run on the robot they are intended for or on a different robot, a so-called Leader robot, with better hardware, more computing power or sensors. ROS can also run on a single robot and the other robots just connect to that robot, or on each robot capable of running ROS, the so-called Standalone robot. Some very simple robots cannot run ROS and do not have a computer attached to run ROS. A Standalone robot will be used as a Leader robot, needed to run drivers and commands wirelessly for these simple robots.

We propose five basic heterogeneous swarm working modes which are described in the following. We provide demonstration videos for all these use case scenarios in (Gânsari and Buiu, 2015). An overall view of these basic use case scenarios is given in Fig. 9 and 10.

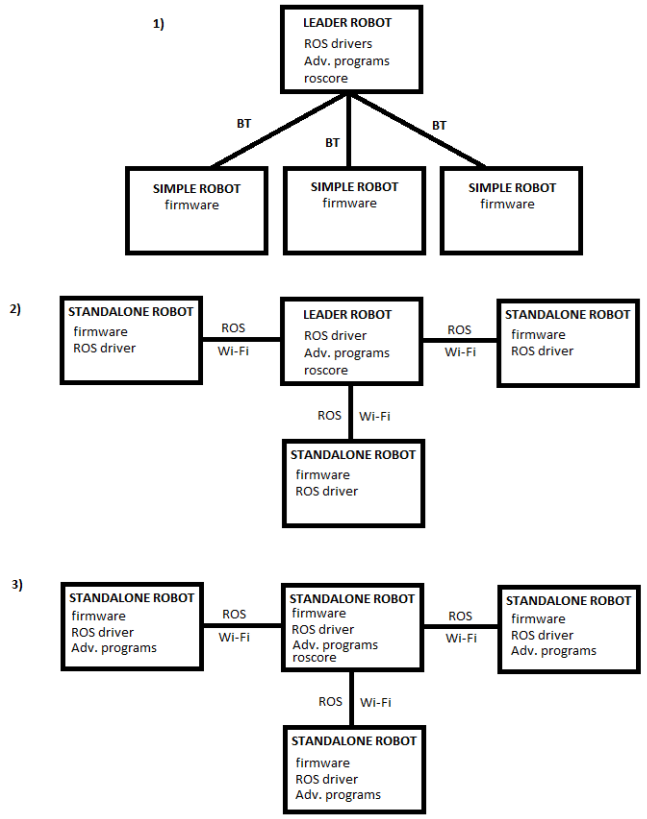

Fig. 9. Heterogeneous swarm functioning modes using ROS (1-3).

1. *Single-Minded Robot Swarm*. In this mode, there is a main robot called the Leader with a computer running the ROS master (roscore), ROS drivers for itself and all the other robots. The Koala robot is the Leader in this system configuration. The Leader sends/receives information to/from the other robots through wireless communication, e.g. Bluetooth (BT) for the robots used here. All the other task-driven programs run on the same main robot computer, which sends commands to the robots through the ROS system. For this mode the Leader robot needs modifications as in section 2.4. This mode is used in order to have a more complex and capable robot control, provide ROS communication and drive the application for a large number of very simple robots.

2. *Swarm with Leader Robot and Standalone Basic Robots*. In this mode each robot runs its own ROS drivers, and other required control programs. They communicate with each other by connecting to the main robot computer which runs the ROS master. The connection is done in a common network using WLAN protocol. The main robot may receive tasks from the user and the other robots will coordinate and receive commands from this Leader robot. For this mode and the following modes, robot modifications as in Section 2.4 are required for the Standalone robots. This mode is recommended when the user wants to control many Standalone robots in the swarm through a single robot. This mode is also recommended when a robot needs to run some programs which require more powerful processing power than is available on its computer, but available on the Leader robot's computer. The processed data is passed over ROS.

3. *Swarm with Standalone Robots Connecting to a Central Robot*. This is similar to mode 2, and the difference is that all the robots act and run individual tasks independent from the other robots, but they communicate using the ROS environment by connecting to a single rosmaster that runs on a designated robot in the network. In this mode all the robots in the swarm are independent, and there is no central controller. The robots just connect to the roscore running on the computer of the one designated robot in the swarm, to allow communication between robots, allowing the robot programs to exchange data.

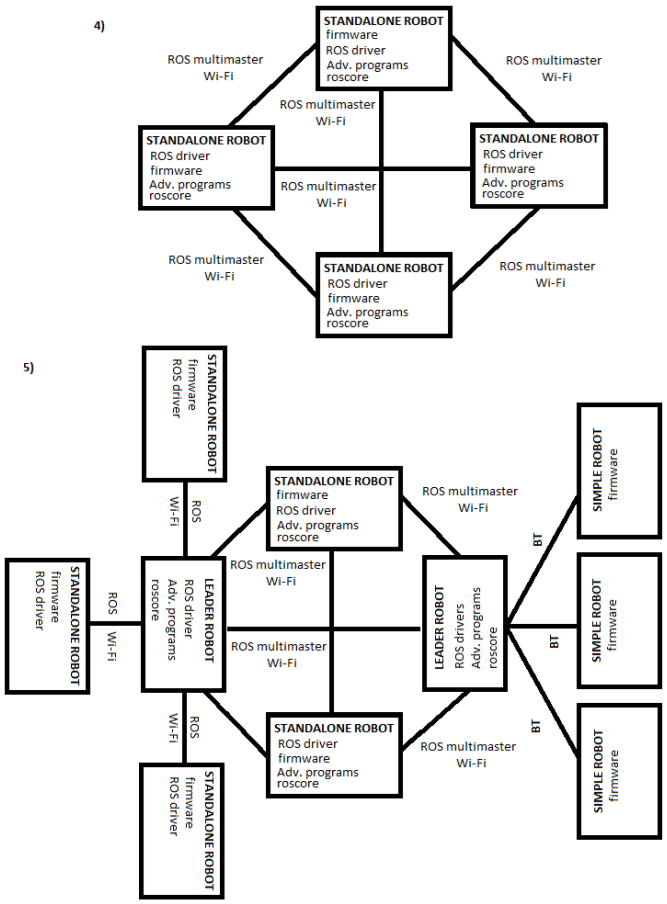

Fig. 10. Heterogeneous swarm functioning modes using ROS (4 and 5).

4. *Swarm with Standalone Robots*. In this mode each robot is a fully independent agent running its own ROS master. The robots connect when they need to other robots by using the rosmultimaster ROS package (http://wiki.ros.org/multimaster_fkie). This package allows the robot to search the network for other ROS masters (individual robots or single-minded swarm of robots in the case of the first mode), synchronize and communicate with each other.

5. *Combined Swarm* (Single-Minded Robot Swarm(s) and Standalone Robots*)*. This is a combination of the previous modes. This mode is used when very simple robots, which lack some of the capabilities required for this system, are added to the swarm. Depending on their hardware capabilities, they can be added to a robot configuration such as in mode 1 or 2. The robots in these configurations can

communicate with other Standalone robots in the swarm, or with similar robots controlled through another Leader robot in a similar configuration.

Fig. 11. Single Minded Robot Swarm.

5. EXPERIMENTAL RESULTS. APPLICATIONS

Using the ROS drivers described in section 3, a series of tests were devised to prove that the robots in the swarm can communicate at a software level using ROS. The robots have individual goals, in this case to follow simple movement commands. The robots move after receiving command information from their program, whether it is local or from another robot. All the programs needed for each test will be called individually, and also roslaunch files can be used to launch all the needed programs. A desktop computer together with the programs SSH and X11VNC were used in the tests to connect remotely to the robots, i.e. the laptop and Raspberry Pis mounted on the robots. Five tests were done to prove each functioning mode.

In the first test, a single-minded swarm with the main robot Koala is running all the drivers and programs for each robot. The programs for the robots communicate through the roscore and send commands to their respective robots using Bluetooth. An example is shown in Fig. 11.

The second test is done with three Standalone robots. Each robot runs its own drivers and they all connect to a single roscore. In this test a Koala robot is used as a Leader running roscore and rtab map program for ROS (http://wiki.ros.org/rtabmap_ros) to create maps with the Kinect sensor. The mapping information is needed for SLAM. To connect to a single roscore, all the robots must have the following two ROS commands run in the terminal to tell the system where the roscore is run and the IP of each robot:

export ROS_MASTER_URI=http://Leader_IP:11311

export ROS_IP=Current_Robot_IP

In the third test all the robots run their drivers and other high-level programs standalone, but they connect and communicate with each other through a single roscore running on a single robot. In this test, the E-puck robot runs along with its driver and uses gmapping ROS nodes (http://wiki.ros.org/gmapping) for SLAM. The K3 is running the driver and a simple movement program. The Koala robot runs rtab map with the Kinect and has a wireless controller connected via USB for movement commands. The controller Linux drivers are linked to a ROS joy node (http://wiki.ros.org/joy) which will publish the controller data. To prove the communication between robots, the controller on the Koala is used to send movement commands to the Koala and E-puck.

In the fourth test each robot is completely independent, running roscore and all programs. The robots connect with each other using rosmultimaster nodes. A detailed report on how to set up the multimaster ROS can be found in (Juan and Coratelo, 2015). All the robots in this test subscribe to the joy node running on the Koala, and thus move by receiving commands from the controller. A multimaster example running on each Standalone robot is given in Fig. 12.

In the fifth and final test a Combined Swarm is tested. All Standalone robots run their own roscore and the rosmultimaster node. The Standalone robots have other simple robots connected to them using Bluetooth and run their own drivers, other more advanced programs and the simple robots' ROS drivers. A tested goal here is to have simple robots such as those from the first test receive commands from other Standalone robots in the swarm through its Leader robot. The Koala controller joy node was

used again, publishing data on the ROS network. Any robot driver connected to this network can subscribe to the publishing joy node and thus the simple robots not connected to the Koala with the controller receive commands from the controller through their own respective Leader robot. A screen capture of this Combined Swarm Robots and the Standalone robots' computer desktops can be seen in Fig. 13.

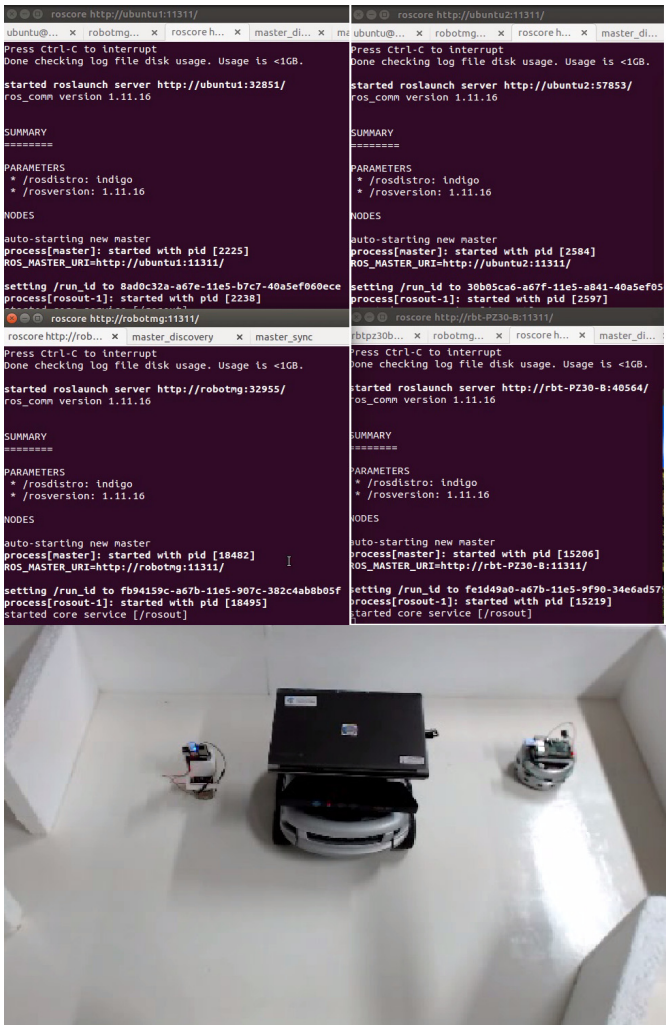

Fig. 12. Multimaster ROS. 4 roscores running on the user computer and 3 remote SSH terminals connected to the computers of the robots. ROS master and sync nodes are running on all computers.

6. CONCLUSIONS

Robotic swarms can move to a superior level of performance and applicability if a large number of robots with different kinds of heterogeneities in terms of characteristics and objectives can be integrated in a coherent whole. With this goal, we presented an original approach to the problem of software integration for swarms of heterogeneous mobile robots. The contributions of our work include: software integration of different robots in a single software environment allowing communication between robots (i.e. robot programs) using add-on hardware and ROS, providing a template for a ROS compatible robot driver, and proposing five basic working modes for heterogeneous robot swarms with functioning examples which are demonstrated through online videos.

Fig. 13. User connected to the heterogeneous swarm. A computer with webcam is used to connect remotely via Wi-Fi to the computer desktops of 3 Standalone robots: Koala+Laptop, K3+Raspberry Pi and E-puck+Raspberry Pi. The other robots are connected via Bluetooth to their respective Leader robots. ROS cores, drivers and applications are running in the background terminals.

The system offers scalability, allowing the addition of any number of robots such as those presented in this paper, and other ROS compatible robots. This system of robots can be used to raise the level of automation in factories and even in places for the general public.

Plans for further developments are manifold and involve several activities. A major activity is having the robots in the swarm accomplish advanced coordinated and individual tasks by using the advantages of the swarm and of the different capabilities of each robot. Such an example is having the robots navigate in an intelligent manner using simultaneous localization and mapping. Another use is to add new different robots to the swarm which will provide new capabilities to the system.

Having robots communicating and coordinating over the network can leave the robots exposed to security threats from other robots and computers that connect to the network, and can drive the robots in a different manner than the intended one. A security system solution for the heterogeneous robot

swarm is required and this can be accomplished by using bioinspired approaches.

ACKNOWLEDGEMENT

This work was supported by the grant of the Ministry of National Education CNCS-UEFISCDI, project number PN-II-ID-PCE-2012-4-0239, Bioinspired techniques for robotic swarms security (Contract 2/30.08.2013).

REFERENCES

Alisher, K., Alexander, K., and Alexandr, B. (2015). Control of the Mobile Robots with ROS in Robotics Courses. *Procedia Eng.*, vol. 100, pp. 1475–1484.

Araújo, A., Portugal, D., Couceiro, M. S., Sales, J., and Rocha, R. P. (2014). Desarrollo de un robot móvil compacto integrado en el middleware ROS. *Rev. Iberoam. Automática e Informática Ind. RIAI*, vol. 11, no. 3, pp. 315–326, Jul. 2014.

Dorigo, M., Floreano, D., Gambardella, L. M., Mondada, F., Nolfi, S., Baaboura, T., Birattari, M., Bonani, M., Brambilla, M., Brutschy, A., Burnier, D., Campo, A., Christensen, A. L., Decugniere, A., Di Caro, G., Ducatelle, F., Ferrante, E., Forster, A., Gonzales, J. M., Guzzi, J., Longchamp, V., Magnenat, S., Mathews, N., Montes de Oca, M., O'Grady, R., Pinciroli, C., Pini, G., Retornaz, P., Roberts, J., Sperati, V., Stirling, T., Stranieri, A., Stutzle, T., Trianni, V., Tuci, E., Turgut, A. E., and Vaussard, F. (2013). Swarmanoid: A Novel Concept for the Study of Heterogeneous Robotic Swarms. *IEEE Robotics and Automation Magazine*, vol. 20 (4), pp. 60–71, Dec. 2013.

Gânsari, M and Buiu, C. (2015). Demonstrative videos for software system integration of hetereogeneous swarms of robots. [Online] *http://dx.doi.org/10.5281/zenodo.35649.*

Juan, S. H. and Coratelo, F. H. (2015). Multi-master ROS systems. *Technical Report IRI-TR-15-1*, Institut de Robòtica i Informàtica Industrial, Barcelona, Spain, Jan. 2015.

K-Team (2012). *Khepera III User Manual*, V3.4, Apr. 2012.

K-Team SA (2001), *Koala User Manual*, V2.0, 29 May 2001.

Mayachita, I., Widyarini, R., Sono, H. R., Ibrahim, A. R., and Adiprawita, W. (2013). Implementation of Entertaining Robot on ROS Framework. *Procedia Technology*, vol. 11, pp. 380–387.

Moon, W.-S., Jang, J.-W., Kim, H.-S., and Baek, K.-R. (2015). Virtual Pheromone Map Building and a Utilization Method for a Multi-purpose Swarm Robot System. *International Journal of Control, Automation and Systems*, vol. 13 (6), pp. 1446–1453.

Patil, M. D. and Abukhalil, T. (2014). Design and Implementation of Heterogeneous Robot Swarm. *ASEE 2014 Zone I Conference,* April 3-5, 2014, University of Bridgeport, Bridgeport, CT, USA.

Pinciroli, C., Lee-Brown, A., and Beltrame, G. (2015). Buzz: An Extensible Programming Language for Self-Organizing Heterogeneous Robot Swarms. *arXiv:1507. 05946*, p. 12, Jul. 2015.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R. and Ng, A. Y. (2009). ROS: an open-source Robot Operating System. *ICRA workshop on open source software*, vol. 3 (3.2), p. 5, 2009.

Sahin, F., Sridhar, P., Horan, B., and Jamshidi, M. (2007). System of systems approach to threat detection and integration of heterogeneous independently operable systems. *IEEE International Conference on Systems, Man and Cybernetics*, pp. 1376–1381, 2007.

Şahin, E. and Spears, W. M. (2005). *Swarm Robotics*, vol. 3342, pp. 10–20, Eds. Springer Berlin Heidelberg.

Tsarouchi, P., Makris, S., Michalos, G., Matthaiakis, A.-S., Chatzigeorgiou, X., Athanasatos, A., Stefos, M., Aivaliotis, P., and Chryssolouris, G. (2015). ROS Based Coordination of Human Robot Cooperative Assembly Tasks-An Industrial Case Study. *Procedia CIRP*, vol. 37, pp. 254–259.

Wei, H., Shao, Z., Huang, Z., Chen, R., Guan, Y., Tan, J., and Shao, Z. (2016). RT-ROS: A real-time ROS architecture on multi-core processors. *Futur. Gener. Comput. Syst.*, vol. 56, pp. 171–178, Mar. 2016.