Multi-robot data mapping simulation by using microsoft robotics developer studio

Antonio Matta-Gómez *, Jaime Del Cerro, Antonio Barrientos

Centre for Automation and Robotics, CAR-UPM-CSIC, Madrid, *Spain*

A B S T R A C T

This document summarizes the goals achieved in the development of a data mapping application, for a multi robot system, implemented as a service with the guidelines found in the Service Oriented Computing paradigm (SOC). The obtained service generates both local and global maps in the reconstruction of a virtual scenario: the local maps represent the surrounding area around each one of the mobile robots, and the global one the totality of the scenario where the robots move. The information of the global map is continuously updated by merging the data coming from the local maps by using a novel approach : each one of the maps manages a confidence level value that defines which of the data coming from the maps is worthy of being updated into the global one. This technique is not present in related work. The *Microsoft Robotics Developer Studio* framework was chosen for its implementation because of the advantages that this tool offers in the management of concurrent and distributed processes, typically found in both a robotics platform and in a multi robot system.

Keywords
Service Oriented Computing (SOC), Multi-robot systems, Data mapping, Robotics frameworks, Microsoft robotics developer studio

1. Introduction

It is advisable to count with a simulation software that allows the study of the behavior of a multi robot system in a virtual scenario previous to their execution in a real one [1]. This simulation software allows the supervision and analysis of the tasks that the multi robot system can perform and offers the capability, to human personnel, to train and control unmanned robots in collaborative environments [2].

These simulators must be consistent with the type of concurrent and distributed processes that are typically found in a robot or in a multi robot system; processes that allow both the execution of multiple tasks concurrently and the communication among them in a distributed approach [3].

This is why the *Microsoft Robotics Developer Studio* (MRDS) framework was chosen as the robotics simulator of this project. It not only offers two powerful dynamics and graphics engines [4], it also allows the implementation of concurrent and distributed processes in a novel approach that it can only be found in this simulator [5].

This approach is possible because the MRDS is built upon two unique libraries: the Concurrency and Coordination Runtime (CCR) and the Distributed Software Services (DSS). They both run as .NET DLLs and offer the possibility to build asynchronous, concurrent, and service oriented applications with different programming languages [4]. These are key elements found both in robotics applications and in a software programming paradigm called Service Oriented Computing (SOC), which sees a software system as a set of services that are hierarchically coordinated and in continuous communication.

* Corresponding author
E-mail address antonio.matta@upm.es (A. Matta-Gómez)

By following this paradigm a data mapping service was obtained. This service was in charge of coordinating and interre lating all the components that were part of the data mapping service, which were also modeled as MRDS services. With the obtained collaborative behavior, it was possible to reproduce the main components of a virtual scenario through the gener ation of accurate maps using a multi robot system.

The generated maps take into consideration information such as obstacles, terrain gradients, traction estimates, quality of the communications, and GPS coverage. These maps are later used in the virtual system for navigation, path planning, and obstacle detection purposes.

Prior to the development of the mapping service, several mapping techniques were studied [6 8]. It was noticed, that most of these techniques divide the information needed for the recreation of the scenario in two separated parts. The first part is a global map where the static elements of the scenario, such as walls or roads, are represented [6]. This map basically allows the robot to perform path planning tasks for long distances or long periods of time. The second part introduces the concept of instant representation, which basically takes the last sensor readings and creates a map local to the robot that allows the robot to react if dynamic obstacles are detected [7].

Moreover, it was found that two different models can be derived from the different existing representations: the geomet rical elements model and the occupancy cells one [8]. The first one is made of a set of geometrical primitives (points, corners, walls, shapes and so forth) whose positions are constantly estimated from the information provided by the sensors data. The second one represents the environment as a set of cells where each one of them holds information related to the scenario.

The occupancy cell model, also known as occupancy grids [9], was used over the geometrical one because it makes easier the fusion of data coming from different sensors readings [10], something that it is constantly performed by the data mapping service in order to obtain better surfaces and correct faulty readings with more recent and accurate ones.

Taking all this into account, a multi layer grid structure was created to represent the maps generated by the robotics mapping service. This structure holds a representation of the scenario in a set of layers, where each layer is divided into a grid of cells that represents an element of the scenario.

The maps are characterized by both the resolution of their information, high or low, and the size of the area of the sce nario that they can represent: local to the robot or global if it contains the entire scenario. To generate these maps, the data mapping service has the following data as input parameters: an initial low resolution 2.5 dimensional global map that rep resents the static data present in the virtual scenario, the position and attitude of each one of the robots, and the 3D laser readings coming from every robot.

With these inputs, the data mapping service generates a global map that contains both the information of the initial global map and the one coming from the local maps generated by each one of the robots. This is achieved through a fusion of information process based on the level of confidence present in the cells of the maps that are being fused, something that has not been found in previous related work, and that was possible because of the features present in Service Oriented Computing paradigm.

This document is organized as follows: Section 2 gives a brief description of the Service Oriented Computing program ming paradigm and the benefits that MRDS services can offer to multi robots applications. In Section 3 the obtained data mapping service is carefully described, as well as the two types of maps that this service generates. Section 4 describes the software architecture of the obtained service. Later, in Section 5 the simulation results obtained through the use of the data mapping service are shown and Section 6 presents the conclusions of this article and the future work related to the mapping techniques described in this document so as to improve it. The advantages of using the *Microsoft Robotics Developer Studio* framework in the implementation of the services are also described.

2. SOC computing and multi-robot systems

Service oriented computing, also known as SOC computing, has become a standard of the software industry in the area of the creation and distribution of services across networks. It is based on object oriented computing (OOC) and the coordination and synchronization of services. Basically, a service is an object that holds a proxy server used to be connected, in a distributed approach, with other services that can be located either in the same machine, in different ones, or in other networks [11].

By this way, developers can easily share services for their apps, which can be programmed by using different languages and different developing teams; allowing in this manner modularity, reliability and easy code reuse [4]. All the main soft ware companies have adopted and supported this new paradigm [5].

This technology could not have been ignored by the robotics software industry. On the contrary, as early as 2005, SOC computing concepts started to be applied in robotics applications [12] and in 2006 Microsoft launched its SOC based robot ics framework called Microsoft Robotics Developer Studio (MRDS) [4].

This framework is a Microsoft Windows® based system focused on the creation of robotics applications that can either be simulated through the use of a three dimensional virtual simulator called Visual Simulated Environment (VSE) or connected to real robots through a programming interface known as Visual Programming Language Environment (VPL).

The Visual Simulated Environment is composed by the XNA graphics engine, an engine that gives such a realistic rendered images that it is used by most of the state of the art video games that run inside the Microsoft XBOX video game machine, and a dynamics engine called AGEIA, widely used for realistic simulations. These both tools turns this framework into an attractive and advantageous one for the three dimensional simulation of multi robot systems [13].

As it was previously mentioned, the two main libraries in which this framework is based on are called Concurrency and Coordination Runtime (CCR) and Distributed Software Services (DSS). They are in charge of both giving concurrent capabil ities to robotics services (CCR) and allowing the creation and communication of services in a distributed approach (DSS).

More specifically, CCR allows multi threading and the synchronization of processes without the use of classical tech niques such as multi threads, mutexes, locks, and semaphores. It is based on structures called *Ports*, which are strongly typed interfaces used by the service both to read incoming messages and to post messages to other services. They can be grouped together into a set of *Ports* called a *PortSet*.

Basically, what happens inside a *Port* or *PortSet* is that after starting to arrive messages, they are queued by functions called *Receivers*. Later, these *Receivers* pass the messages to entities called *Arbiters*, which are in charge of associating each message to its respective function stored in a queue of tasks called the *Dispatcher Queue*. From there, they are sent for exe cution to a node of threads called the *Dispather*.

Most of the time CCR handles all this transparently, so there is no need to define mutexes or write callback procedures as it is done in classical multi threading programming.

On the other hand, the DSS library is built on top the of the CCR and it is used to create services or distributed services applications. A DSS application is a set of multiple independent services running in parallel. In a robotics application, these services could be hardware components such as sensors and actuators, or software entities as user interfaces or a dynamics simulation engine, as it is the case of the MRDS. All of them are grouped together and programmed to share messages between them. Moreover, these services can be operating in a same hosting environment, called a DSS node, or distributed over a network, allowing the flexibility for execution of computational expensive services in distributed computers [12].

The main components of a service are: its identifier, called *Contract Identifier*, it is used to identify a service from other services; its *State*, which holds the current contents of a service such as its attributes and functions; the *MainPort*, or operations port, it is a CCR *PortSet* where messages coming from other services arrive; the *Service Forwarder Port*, also a CCR *PortSet*, it is responsible for the partnering of services running in remote nodes; and finally, the *Service Partners* that enable services to be combined, or composed, as partners to create applications; a process that in MRDS terminology is known as orchestration. Fig. 1 shows a graphic representation of these concepts in a DSS architecture.

The sending and receiving of messages are accomplished by using the service **Main Port**. This Proxy holds two key functions called *Get* and *Post*, which are in charge of receiving and sending messages, respectively.

Noticing all the advantages that SOC computing offers to the development of robotics applications and which are present in the MRDS, this framework was chosen to built a virtual simulator where a set of mobile robots maps its surrounding environment.

The multi robot system uses the CCR and DSS libraries extensively to implement the concurrent and distributed tasks needed by the robots to map the system in a collaborative way. But first, it will be explained how the mapping service works, and later how it was implemented using the MRDS framework.

3. The robotics data mapping service

In the obtained mapping system, a group of three robots collaborates in updating the information stored in a global map, called the Global Low Resolution Map (**GLRM**), which contains a 2.5D representation of all the environment where the robots move. In order to fulfill this task, each robot updates a local and individual map called High Resolution Map (**HRM**) that represents the area around the robot, and then shares its updates to the **GLRM** map.

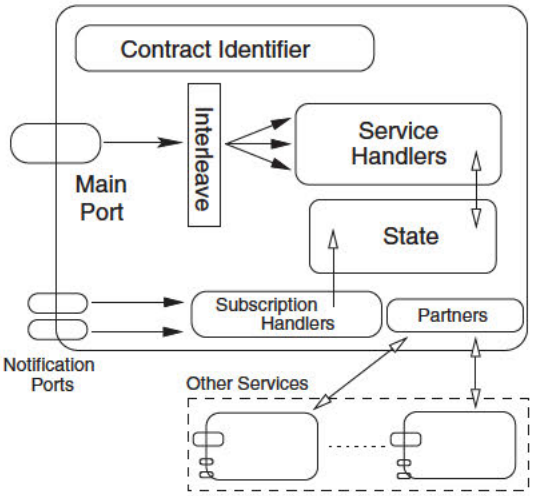

Fig. 1. DSS description and connection among services. Services could be located in the same host or other hosts across networks.

Fig. 2. Virtual Scenario used for testing the multi-robot system.

Fig. 2 shows a global view of the obtained virtual scenario that was used to test the data mapping service. Basically, it is composed of a set of streets, different types of houses, and three Summit XL robots, which are coordinated from a central station and circulate in a patrol mission. It was created using the Visual Simulation Environment of the MRDS framework.

3.1. Data mapping internal processes

Generally speaking, the data mapping service is composed of the following tasks.

First, external information arrives as inputs of an internal data mapping service, present in each one of the mobile robots, which is in charge of the creation of the robot's maps. Among these data are: an static 2.5D global map, the position and attitude of the robot, and the 3D Laser Range Finder readings.

After obtaining all these parameters, the readings of the 3D Laser Range Finder are processed. With this data, the 3D coor dinates of the points of the space with which the laser rays collide can be obtained. These points are called *3D collision points*, which are represented as red dots in Fig. 3.

The collision points are used to create the layers of an auxiliary local map called **3DLaserAuxMap** that later it is fused with the mobile robot's internal **HRM** map. After that, the scale of this updated **HRM** map is modified to obtain a new map, called Local Low Resolution Map (**LLRM**), that holds a resolution that fits with the resolution of the **GLRM** map allowing in this manner to merge the data stored in these two last maps.

The merging of all the **LLRM** maps into the **GLRM** map is performed by one of the services of the simulation system that is called the Map Manager. But not all the data of the **LLRM** maps is sent to the Map Manager. Instead, each one of the robot's internal data mapping service selects the cells that are worthy of being merged and sends them to the Map Manager. How these cells are selected is based on a process that it is carefully described later in this section.

In this way, less information is required to keep the **GLRM** up to date, improving previous work such as [8,14] in terms of memory management and efficiency of data flow among robots.

Once the **GLRM** is updated, the Map Manager service sends the updated cells of the new **GLRM** to each one of the mobile robots and the whole process starts again.

3.2. Structure of the maps

The structure of both the **GLRM** and the **HRM** maps is based on an architecture of layers and cells, where several layers are used for classifying and storing the information associated with the virtual scenario. Each layer stores a different type of

Fig. 3. 3D collision points. Representation of what the 3D laser is scanning at a given moment.

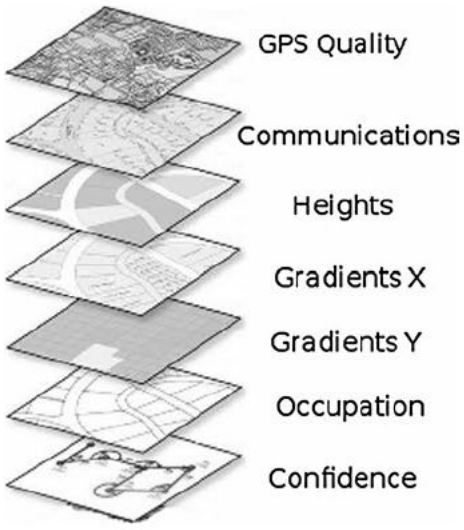

Fig. 4. Layers and cells architecture.

information that defines a feature found on the scenario and it is divided into a set of cells where each cell represents an area of the surface as it is found in an occupancy grid map [6]. Fig. 4 shows the layers that conform these maps.

How these layers are composed and generated by the data mapping service is carefully described in the following sub section. It is worth noting that these layers are not only used by the robots for mapping purposes, but also for navigation, path planning and obstacle detection ones.

The differences between the **GLRM** and the **HRM** maps resides in both the actual physical space that each one of them can represent and the size of the area of the cells that conforms the resolution of the information. Basically, the low resolution maps represent the information of the whole simulation environment, while the high resolution ones possess detailed infor mation of the nearby zone that surrounds each one of the mobile robots.

To accomplish this, the area of the cells of the **GLRM** is big, thus the resolution of their contents is low, and in the case of the cells of a **HRM** map, their area is small, hence they store data in a resolution considered to be as high.

3.3. Layers reconstruction process

Inside the data mapping service that runs in each one of the robots, two maps are created: the HRM map and the 3DLase rAuxMap. With these two maps, it is possible to represent the local area present around a given robot.

The HRM is initialized with information coming from the GLRM map and the 3DLaserAuxMap is generated using the last 3D laser scan that arrives from the 3D Laser Range Finder.

Both the information of the **GPS Quality** and the **Communications** layers of these maps is not computed, but given as part of the configuration of the scenario when the simulation starts. The first layer defines the type of GPS signal that could be available on a given area of the virtual scenario. If its value is 1, denotes a single GPS signal, and if its value is 2, a differential GPS signal is found. Similarly, the second layer states the type of communications present among robots: 0 if there is no com munication signal and 1 if there are communications. This value is associated with the interconnection capability present among robots or between a robot and the base station.

As it was previously explained, the readings coming from the 3D laser are processed and a list of three dimensional col lision points is obtained. With this list, it is possible to find out which cell of the map a collision point belongs to. The cells of the third layer, the **Heights** layer, stores a numerical value that represents the average height of all the observed 3D collision points that belong to a given cell. This value defines how high and object can be, regardless if it is a dynamic or static one, something very useful for object detection tasks.

The method used to compute the average height comes from [15], which is based on estimating the best plane that approximates the cloud of points that exists on every cell of the map. The height of the center of that plane is the average height of all the collision points present in the cell and it represents the height of that cell. These heights are always calcu lated using its absolute value, referencing every collision point with the fixed reference system.

The **Gradients in *X* and *Y* direction layers** store the estimation of the terrain's slopes for each cell according to the ori entation given by the *X* and *Y* axes. These slopes define the difficulty that a robot may find while moving through a cell and they are always calculated based on fixed axes. Consequently, the *X* gradient measures how the height varies when the robot is moving along the fixed *X* axes. Similarly, the *Y* gradient measures how the height varies when the robot is moving along the fixed *Y* axes.

The above was very useful to help the robots navigate around the scenario: by using these gradients values, the robots could easily determine which sectors could overstep and which ones not.

To compute the gradients, the following procedure is applied.

The equation of the plane is expressed in the form of:

$$Z \quad a * X + b * Y + c \tag{1}$$

where Z is the height of the cell and X and Y are the coordinates located in the plane of the map. The gradient according to X is $G_x \quad \frac{\delta Z}{\delta X} \quad a$ and the gradient according Y is $G_y \quad \frac{\delta Z}{\delta Y} \quad b$. Therefore, the gradients can be calculated by finding the coefficients of the plane equation.

Hence, the task is to find the hyper plane that best fits this plane equation. One criterion to do it is to minimize the sum of squared errors according to the following expressions.

The model of multiple regressions is expressed by:

$$Y_i \quad \beta_0 + \beta_1 * X_{1i} + \beta_2 * X_{2i} + \cdots + \beta_k * X_{ki} + u_i \tag{2}$$

where $u_i \rightarrow N(0, \sigma^2)$ and $\beta_0, \beta_1, \beta_2, \ldots \beta_k, \sigma^2$ are unknown parameters to be computed.

If we expressed it in matrix form:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad \begin{bmatrix} 1 & x_{11} & x_{21} & \cdots & x_{k1} \\ 1 & x_{11} & x_{21} & \cdots & x_{k1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{1n} & x_{2n} & \cdots & x_{kn} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{bmatrix} + \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} \tag{3}$$

$$Y \quad X\beta + U$$

$$U \rightarrow N(0, \sigma^2 I)$$

For the case that concern us, the vector of dependent variables is the vector of heights found on every cell, and the rest of independent variables are the X and Y coordinates.

To estimate the following system using the sum of squared errors:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad \begin{bmatrix} 1 & x_{11} & x_{21} & \cdots & x_{k1} \\ 1 & x_{11} & x_{21} & \cdots & x_{k1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{1n} & x_{2n} & \cdots & x_{kn} \end{bmatrix} \begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \vdots \\ \hat{\beta}_k \end{bmatrix} + \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} \tag{4}$$

$$Y \quad X\hat{\beta} + e$$

where the vector e is defined by:

$$\|e\|^2 \quad \sum_1^n e_i^2 \tag{5}$$

In order to e^2 be a minimum, e has to be perpendicular to the vector space generated by the columns of X.

$$X \quad \begin{bmatrix} 1 & x_{11} & x_{21} & \cdots & x_{k1} \\ 1 & x_{11} & x_{21} & \cdots & x_{k1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{1n} & x_{2n} & \cdots & x_{kn} \end{bmatrix}, \quad e \quad \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} \tag{6}$$

Fulfilling that:

$$\begin{aligned} &X^T e \quad 0 \\ &X^T Y \quad X^T X\hat{\beta} + X^T e \\ &X^T Y \quad X^T X\hat{\beta} \rightarrow \hat{\beta} \quad (X^T X)^{\ 1} X^T Y \end{aligned} \tag{7}$$

and the values of β_1 and β_2 correspond to the gradients according to X and Y respectively.

For this process, it is necessary that at least three points drop in the same cell in order to estimate the hyper plane. This does not necessarily occur on every cell of the map. Therefore, the collision points found in adjacent cells are used to calcu late the linear regression plane for those types of cells, as it is shown in the following equations.

$$GX_{ij} \quad \frac{Z_{j+1} \quad Z_j}{Scale} \tag{8}$$

$$GY_{ij} \quad \frac{Z_i \quad Z_{i\ 1}}{Scale} \tag{9}$$

where

- GX_{ij} is the gradient referenced to the X axis of the (i,j) element.
- GX_{ij} is the gradient referenced to the Y axis of the (i,j) element.
- *Scale* is the value of the side of the cell.

This method, which is called *calculus of the gradient using rows and columns*, offers approximated results, but not very accurate ones, and the gradients obtained are not centered in the position of the cell where they belong, but they are a little bit displaced.

To improve the results obtained with this method, four adjacent cells were used simultaneously to calculate the gradients both in the X and Y directions using the heights value from each one of those four adjacent cells.

The **occupation probability layer** defines how busy a cell may be according to the slope found in that cell. Therefore, if the slope of the cell is greater than the one tolerated by the mechanical capabilities of the robot, the terrain will be considered as not passable and if, for the contrary, the absolute value of the slope is permissible for the robot, the level of occupation will be a numerical value that linearly rises with the absolute value of the slope. This layer is very useful for path planning purposes: if various paths are generated, the least difficult one is selected depending on the values found in this layer [16].

To estimate the slope of the cell, the previously computed gradients values are used. These gradients represent the tan gent according to the X and Y directions present on every cell.

Therefore, it was decided to give a value of one (busy) to those cells with a gradient value of 0.3, which represents the maximum slope that a robot can overpass in our simulation.

For those cells whose gradient value is inferior, the occupation is estimated through a linear function that increases the occupation state from a null gradient to the highest admissible one. Although this method is very simple, it turned out to be very useful for path planning purposes.

Finally, the **confidence layer** is used to estimate how reliable the data stored in each cell is. The numerical values gath ered in this layer varies in a range that goes in the [0,255] interval, in such a way that 0 stands for a total and 255 means that the information about that cell is completely reliable. The process starts with an initial confidence value inherited from the GLRM map, and it gradually stabilizes as the mobile robot performs the exploration of the environment.

This confidence value is based on both the number of collision points present in a given cell and the amount of time elapsed since the last reconstruction of the map because the mapping process deals with dynamic scenarios.

Because of the shape of an scenario and the position of the robot on it, it may occur that some cells may have many impact points while their nearby cells may have very little or even none impact points at all. Besides, and depending on the size of the cells, it is possible that all the impact points may be concentrated in a small sector of a cell while the rest of the cell remains empty. For these reasons, the confidence value found in every cell is related not only with the number of the impact points kept in a given cell, but also with the dispersion present in all of their respective areas.

To compute the confidence value, every cell was divided in sixteen sub cells, as it is shown in Fig. 5.

The algorithm of confidence assignment estimates the associated sub cell of each impact point through the list of impact points belonging to a cell and their respective coordinates. Thus, the sub cells where there is at least one collision point are counted. The confidence is then calculated by dividing the number of sub cells, with collision points, and the total number of sub cells n. During the test phase of this algorithm, it was decided to give the value of sixteen to n, and it was also found that the wider the area of the cell the bigger this number must be.

After applying the division, a value between 0 and 1 that represents the confidence of the cell is obtained.

With this procedure, the distribution among the impact points is considered and the confidence based on that distribu tion obtained. Besides this criterion, other parameters as the distances of the 3D Laser Range Finder scan to the impact points have to be considered.

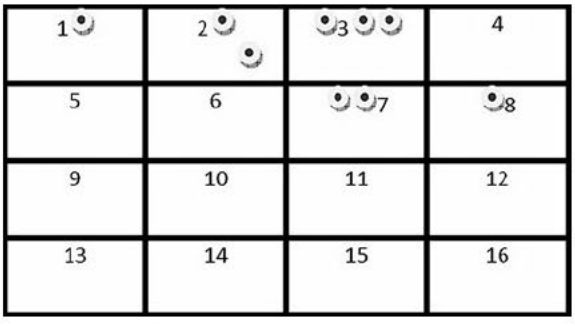

Fig. 5. Sub-cells division.

Because of the errors found in the measurements provided by the 3D Laser Range Finder and the estimation of the position and orientation of the mobile robot, the errors generated while measuring the distances raise when the collision distance increases. Therefore, it is useful to penalize the confidence of far distance collision points in order to add a certain uncertainty over them. To accomplish this, the following correction criterion was applied:

For small distance measures, less than 75% of the maximum range, a low penalty value is applied, and for long distances measures, more than 75% of the maximum range, a high penalty value is applied.

The correction coefficient shown in Fig. 6 can be mathematically expressed using Eq. (10).

$$f \quad \begin{cases} \frac{0.2D}{0.75RangoMax} + 1, & \text{if } D < 75\% \text{ of max range} \\ \frac{0.2(D \quad 0.75RangoMax)}{0.20RangoMax}, & \text{if } D > 75\% \text{ of max range} \end{cases} \tag{10}$$

The confidence calculated remains invariable unchanging for distances closed to the mobile robot and it is reduced for distances that are far away from the robot.

Once the layers of the 3DLaserAuxMap are obtained, they are merged with the layers of the HRM map. A process that is described in the following sub section.

3.4. Maps merging

There are two cases of fusion of information between the layers of two maps:

- Between the 3DLaserAuxMap and a HRM map, something that it is executed in the internal data mapping service present in every robot.
- Between all the updated cells generated by the mobile robots and the global GLRM map. A process that it is coordinated by the Map Manager service.

In the first one, the algorithm in charge of merging these two maps starts verifying the confidence layer of the 3DLase rAuxMap. Those cells that have a not null confidence value state that there is new information in them since the last scan ning, so it is needed to merge that information with the respective cells of the HRM map. In this process, two possible cases can be identified:

- The data stored in the cells that are being merged are similar.
- The data are completely different.

To decide if the data being compared is similar, a threshold of ±10° between the gradients of the cells was established.

If the information is similar, the cells are fused by applying the following weighting criterion, which is based on the con fidence level present on the cells that are being merged:

$$data_{k+1} \quad \frac{data_k * Conf_k + data_{3DLaserMap} * Conf_{3DLaserMap}}{Conf_{ponderation}} \tag{11}$$

where $data_{k+1}$ represents the cell obtained after the merging is done, $data_k$ the cell coming from the HRM map, and $data_{3DLaserMap}$ the one that comes from the 3DLaserAuxMap. The term $Conf_{ponderation}$ contains the sum between the confidence values of the two cells that are being fused.

If the data it is not similar, the following possibilities are considered:

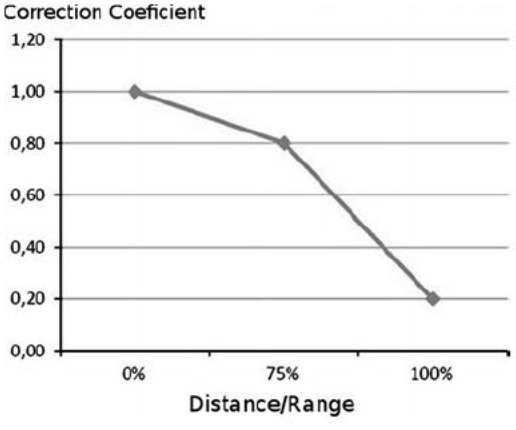

Fig. 6. Correction coefficient in function of the measured distances and the sensor range.

- The difference between the confidence levels is high. This means that one of the two values is more reliable than the other, the cell with the highest confidence value is chosen.
- The difference is low and the confidence values are also low, the cell with the highest confidence value is chosen.
- The difference is low but the confidence values are high. In this situation, it is probable that there has been a change in the environment. Because of this, the new estimation substitutes the old one.

It is considered that The difference between the confidence levels is high if the following expression is satisfied:

$$Conf_{ponderation} > \left(\frac{2}{16} + Conf_k\right) \tag{12}$$

With this process, the information contained in these two high resolution maps is merged and a new HRM map is obtained.

Concurrently, a list called *UpdatedCellsList* is created and filled with only the cells that have been merged between the 3DLaserAuxMap and HRM map. Later, this list is sent to the Map Manager service where the second case of maps merging occurs. In this service, all the *UpdatedCellsList* coming from all the mobile robots are fused together with the GLRM map.

As a result, an updated GLRM map is obtained and the Map Manager service sends the updated cells of this up to date global map to each one of the mobile robots.

When a new 3D Laser Range Finder reading arrives to the mobile robot, the whole process is repeated again.

4. Software architecture of the data mapping service

It was not difficult to translate the processes involved in the creation of the maps to a software architecture based on the SOC computing approach of the MRDS. Basically, the key components were designed and implemented as MRDS services and almost all of the services were interconnected among themselves.

To simulate the communication among services, a set of interface classes were defined for data exchange among services, associating each interface to the type of message(s) that a service could receive or communicate.

Therefore, a fully service oriented application was obtained where all its services were implemented on top of the *CCR* and *DSS* libraries. This gave full SOC capabilities to the implemented services such as: services collaboration, standardized ser vices description, and distributed simulation.

Fig. 7 shows the overall service structure. Each rounded box stands for a service of the system. Services that are created more than once are grouped by a dash line. The arrowed lines between the service icons symbolize an interface between two services and the squared boxes represent the message shared between the services.

Orchestration services were used for the organization of the services structure following a bottom up approach. The main orchestration service was named as the *Mapping Orchestration Service*, which synchronizes and coordinates all the services that are part of the data mapping simulation process described in the previous section. Thanks to the SOC computing features, this orchestration service is able to manage distributed services that help to keep the information of the GLRM up to date, and thus the elimination of wrong information that can surge in the global process.

Additional to the *Mapping Orchestration Service*, another orchestration service, called the *Robot Orchestration Service*, was implemented. This service was in charge of coordinating the services running inside each robot. Among these services are: a *3D Laser* service that simulates the 3D Laser Range Finder attached to each robot and the *Robot Mapping* service that generates both the robot's HRM local map and the updated cells that later are merged to update the GLRM.

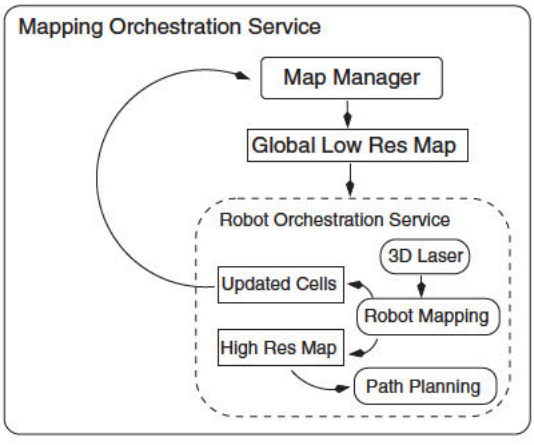

Fig. 7. Services structure of the data mapping process.

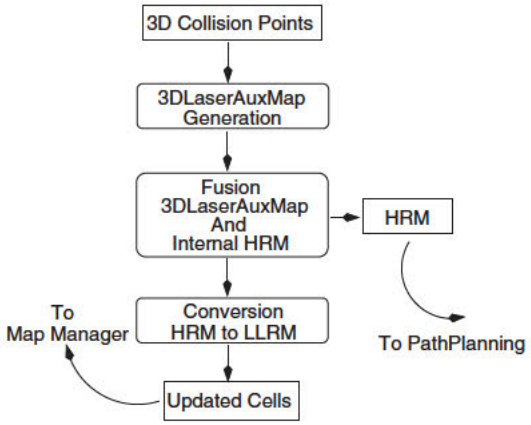

Fig. 8. Mapping processes of the RobotMapping service.

Fig. 8 shows in more detail the internal process that run inside the *Robot Mapping* service that includes the creation of the maps and their respective layers such as it was described in Section 3.

Additionally, as Fig. 7 shows, the updated cells generated by each robot are sent to a service called *Map Manager*. It is in this service where all the major fusion operations take place in order to update the GLRM, such as it was explained in the previous section.

The *Robot Orchestration Service* also models the internal sensors and processes of the robotics platform used in this project. This was the Summit XL mobile robot, a very advantageous one for outdoors exploration [17]. Fig. 9 shows both a real image of this platform and its simulated version inside the virtual scenario.

These internal sensors and processes, such as the navigation sensor fusion or the path planning ones, were also implemented as MRDS services in the same way as the *Robot Mapping* service. This gave the benefit of producing concurrent and distributed modules from scratch, which are continuously and synchronously communicating among themselves. Something that it was only possible to achieve by using the SOC computing paradigm.

Fig. 10 shows, in more detail, the different services that compose the *Robot Orchestration Service*.

All the services developed for this application were entirely developed by the authors of this project and do no exist in any other MRDS services repository. Hopefully, this will be achieved with the help of the services maintainers of the MRDS or shared to other MRDS services repositories such as the ones developed by [11] or [18].

5. Tests and results

The whole data mapping process encapsulated in the *Mapping Orchestration Service* was designed and implemented taking into consideration the *Microsoft Robotics Developer Studio* software development approach. All the services were implemented in the C# programming language using the Integrated Development Environment (IDE) of the *Microsoft Visual Studio* development tool.

For this test, the size of the cell of the GLRM map was set to five meters and the number of cells used to represent the scenario was defined in 200 cells horizontally and 200 vertically. Thus, this map encloses a total area of 1000.000 m^2.

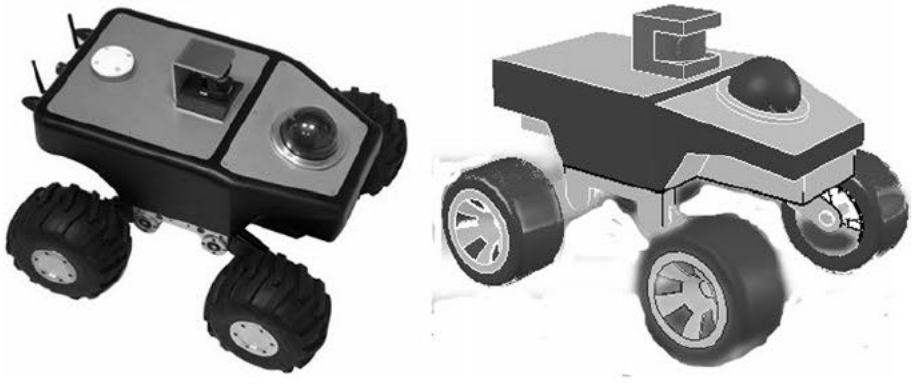

Fig. 9. The Summit-XL platform. On the left, a view of the real robot. On the right, the simulated version of the same robot.

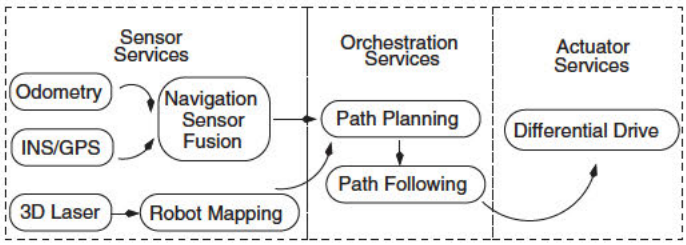

Fig. 10. Schema of the main DSS services that conforms the Summit-XL robot.

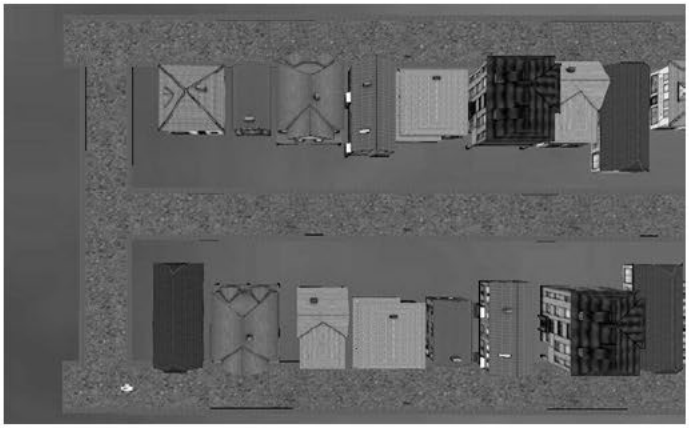

Fig. 11. Local area of the first robot.

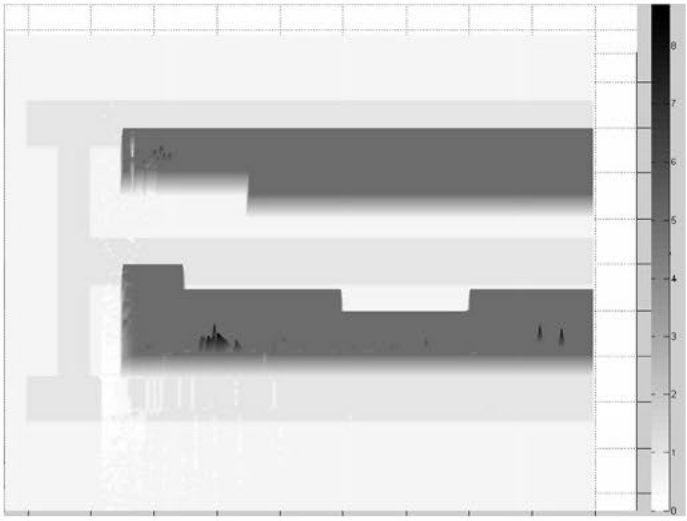

Fig. 12. Initial heights layer of the high resolution map.

Similarly, the high resolution maps were defined by cells with an area of 0.5 meters with a total number of 320 cells horizontally and 320 cells vertically. Therefore, the area of these maps is of 25.600 m^2.

In this section, only examples of the results obtained for the heights and occupation layers are shown, since they are the most representative layers and they are also used for object detection and path planning purposes, respectively.

Fig. 11 shows the surrounding area of the first robot, identified as *robot_001*, which is located in the lower left corner of the scenario.

Fig. 13. Heights layer of the high resolution map after some steps of the simulation.

Fig. 12 shows the obtained initial heights layer of the HRM map of the same robot. It can be seen how the heights of the buildings are preserved and a 2.5D figure is obtained.

Fig. 13 shows the heights layer of the HRM map belonging to the same robot after several steps of the simulation have happened. This layer is obtained after the data coming from the auxiliary 3DLaserAuxMap is fused. It can be seen how the heights values of the buildings have varied into more accurate ones.

Now, for the total scenario, Fig. 14 shows the initial heights layer of the GLRM map as it arrives to the *Robot Mapping* service. It can be seen that the height of the buildings is ten, which is the initial value of the height of all the buildings.

Fig. 14. Initial heights found in the global low resolution map.

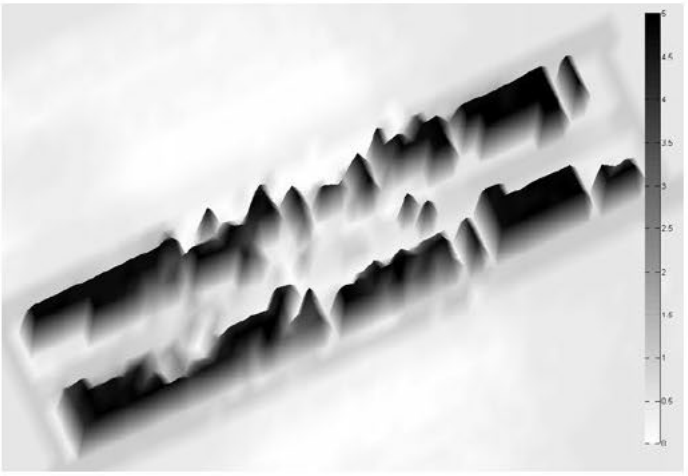

Fig. 15. Heights values at the end of the simulation process.

Table 1
PC-Machine characteristics were the tests were performed.

Characteristic	Property
CPU	Intel Core i3 2.20 GHz
RAM	4.0 GB
Graphics card	Inter HD graphics card
Operative system	Microsoft seven home edition

Table 2
Execution times for the main data mapping algorithms in ms.

Algorithm	*robot_001*	*robot_002*	*robot_003*
Collision point list	55	55	70
Heights layer	5	5	5
Gradient estimation	130	140	307
Confidence layer	82	60	70
Occupation layer	30	30	35
Fusion 3D laser aux map with HRM	5	1	5
Fusion HRM2LRML	110	105	110
Fusion LRML with LRMG	4	5	5

Table 3
Execution times of the simulation process.

Mapping robots	Mapping time
1	445
2	310
3	190

Fig. 15 shows that those height values have been modified to figures that are more in agreement with the real heights of the buildings.

This test was performed in a laptop machine with the characteristics described in Table 1.

The execution time of each one of the algorithms present in the *Robot Mapping* service of each one of the Summit XL robots is shown in Table 2.

Finally, Table 3 shows the time of the simulation process using a different number of robots. It can be seen how the total time used to update the entire global map scales down if more robots are used in the process. Times are in seconds, simulation time.

6. Conclusions and future work

This paper introduces an orchestrated data mapping service, based on SOC computing, that maps the information present in a virtual scenario and that it is used by a multi robot system.

The fusion algorithms that merge the data coming from the different types of maps generated by the data mapping service have been successfully verified. All of them have proven to produce the expected results and run within acceptable execution times.

Our proposed multi layer structure for the maps gives the advantage that more information about the scenario can be stored in the maps, something that enables the map to be used for different types of tasks like navigation sensor fusion, path planning, obstacle avoidance, and grid based localization.

In addition to this, the use of a confidence value for map merging purposes has given accurate and fast results to merge information between maps. This is a technique that we have not found present in previous related work and that it deserves to be studied more carefully in future projects.

Employing the *Microsoft Robotics Developer Studio* as the framework for implementing the data mapping service gave a priceless opportunity to test the benefits that the SOC computing paradigm can give to multi robots applications. Both the *Concurrency and Coordination Runtime (CCR)* and the *Decentralized Software Services (DSS)* introduce a novel approach for the creation and maintenance of concurrent and distributed processes, which obviously are mandatory needed in robotics applications, and allow the creation of concurrent and distributed software from the start.

Although the data mapping works correctly when it is used with a single robot or a small set of robots, it is still unknown how it will work if it is used by a large set of robots. In order to clear this doubt, further tests must be made with the *Microsoft Robotics Developer Studio* framework or even another framework to realize if the obtained module complies with more complex patrol missions.

Finally, tests with real robots are needed as a way to know how this module behaves within a real scenario. This can be accomplished by using the *Microsoft Visual Programming Language (VPL)* environment, included in the MRDS, and that it is used to transfer the developed algorithms to real robotics platforms that supports the *Microsoft Robotics Developer Studio*.

This is an ongoing process that still needs to be tested and that will also give the opportunity to use the obtained data mapping from the web, either as a web service [19] or from a robots as a service cloud [20].

Acknowledgments

This work has been supported by the Robotics and Cybernetics Research Group at Technique University of Madrid (Spain), and funded under the projects ROTOS: multi robot system for outdoor infrastructures protection, sponsored by Spain Ministry of Education and Science (DPI2010 17998).

Appendix A. Supplementary material

Supplementary data associated with this article can be found, in the online version, at http://dx.doi.org/10.1016/j.simpat.2014.10.003.

References

[1] E. Colon, Survey of software frameworks for robotics applications technical report, in: N.S. Organization, Technology (Eds.), Multi-Robot Systems in Military Domains, chap. Annex G, ISBN 978-92-837-0043-2, 2008, pp. 1–32 <http://www.cso.nato.int/pubs/rdp.asp?RDP=RTO-TR-IST-032>.

[2] E. Colon, Robotics frameworks, in: Daisuke Chugo, Sho Yokota (Eds.), Introduction to Modern Robotics II, iConcept Press Ltd., ISBN 978-1463789442, 2013 (chap. 1) <http://www.iconceptpress.com>.

[3] A. Cesetti, C. Scotti, G. Di Buo, S. Longhi, A service oriented architecture supporting an autonomous mobile robot for industrial applications, in: 18th Mediterranean Conference on Control & Automation, IEEE, 2010.

[4] J. Jackson, Microsoft robotics studio: a technical introduction, Robotics & Automation Magazine, IEEE (December) (2007) 82–87. <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4437755>.

[5] J. Cepeda, L. Chaimowicz, R. Soto, Exploring microsoft robotics studio as a mechanism for service-oriented robotics, in: Robotics Symposium and Intelligent Robotic Meeting (LARS), 2010 Latin American, IEEE, 2010.

[6] J.M. Cañas, V. Matellán, Dynamic gridmaps: comparing building techniques, Math. Soft Comput. (2008) 1–8. <http://ic.ugr.es/Mathware/index.php/Mathware/article/view/61>.

[7] S. Thrun, Robotic mapping: a survey, in: Exploring Artificial Intelligence in the New Millenium, Morgan Kaufmann, 2002.

[8] A. Birk, S. Carpin, Merging occupancy grid maps from multiple robots, Proc. IEEE 94 (7) <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1677951>.

[9] A.A.S. Souza, R.S. Maia, L.M.G. Goncalves, 3D probabilistc occupancy grid to robotic mapping with stereo vision, in: A. Bhatti (Ed.), Current Advancements in Stereo Vision, InTech, 2012 (chap. 9).

[10] S. Carpin, Fast and accurate map merging for multi-robot systems, Auton. Robots 25 (3) (2008) 305–316. ISSN 0929-5593 <http://dx.doi.org/10.1007/s10514-008-9097-4>.

[11] W. Tsai, S. Sun, Q. Huang, E. Karatza, An ontology-based collaborative service-oriented simulation framework with microsoft robotics studio, Simul. Model. Pract. Theory 16 (9) (2008).

[12] Y. Chen, X. Bai, On robotics applications in service-oriented architecture, in: 2008 The 28th International Conference on Distributed Computing Systems Workshops, 2008, pp. 551–556, http://dx.doi.org/10.1109/ICDCS.Workshops.2008.60 <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4577843>.

[13] A. Staranowicz, G.L. Mariottini, A survey and comparison of commercial and open-source robotic simulator software, in: Proceedings of the 4th International Conference on PErvasive Technologies Related to Assistive Environments – PETRA '11, 2011, http://dx.doi.org/10.1145/2141622.2141689 <http://dl.acm.org/citation.cfm?doid=2141622.2141689>.
[14] D. Fox, J. Ko, K. Konolige, B. Limketkai, D. Schulz, S. Benjamin, Distributed multirobot exploration and mapping, Proc. IEEE 94 (7) (2006) 1325–1339. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1677947>.
[15] L. Neumann, B. Csébfalvi, Gradient estimation in volume data using 4D linear regression, in: M.G. Hopgood, F.R.A. (Eds.), Eurographics 2000, vol. 19, The Eurographics Association and Blackwell Publishers, Malden, MA, USA, 2000 <http://onlinelibrary.wiley.com/doi/10.1111/1467-8659.00427/abstract>.
[16] S. Opfer, H. Skubch, K. Geihs, Cooperative path planning for multi-robot systems in dynamic domains, in: J. Bedkowski (Ed.), cdn.intechopen.com, InTech, 2011, pp. 237–258 (chap. 11).
[17] Robotnik, Summit XL: A description <http://www.robotnik.es/en/products/mobile-robots/summit-xl> (accessed 30.03.14).
[18] Y. Chen, Z. Du, M. Garcia-Acosta, Robot as a service in cloud computing, in: Fifth IEEE International Symposium on Service Oriented System Engineering, IEEE, 2010.
[19] S. Osentoski, G. Jay, C. Crick, B. Pitzer, C. DuHadway, O. Jenkins, Robots as web services: reproducible experimentation and application development using rosjs, in: 2011 IEEE International Conference on Robotics and Automation (ICRA), 2011, pp. 6078–6083, http://dx.doi.org/10.1109/ICRA.2011.5980464, ISSN 1050-4729.
[20] A. Kouba, A service-oriented architecture for virtualizing robots in robot-as-a-service clouds, in: ARCS 2014, Springer, 2014.