Computational Intelligence, Volume 59, Number 000, 2010

An Ambient Intelligence Approach for Learning in Smart Robotic Environments

DAVIDE BACCIU

Università di Pisa, Italy

MAURIZIO DI ROCCO

Örebro Universitet, Sweden

MAURO DRAGONE

Heriot-Watt University, Edinburgh, UK

CLAUDIO GALLICCHIO

Università di Pisa, Italy

ALESSIO MICHELI

Università di Pisa, Italy

ALESSANDRO SAFFIOTTI

Örebro Universitet, Sweden

Smart robotic environments combine traditional (ambient) sensing devices and mobile robots. This combination extends the type of applications that can be considered, reduces their complexity, and enhances the individual values of the devices involved by enabling new services that cannot be performed by a single device. In order to reduce the amount of preparation and pre-programming required for their deployment in real world applications, it is important to make these systems self-learning, self-configuring, and self-adapting. The solution presented in this paper is based upon a type of compositional adaptation where (possibly multiple) plans of actions are created through planning and involve the activation of pre-existing capabilities. All the devices in the smart environment participate in a pervasive learning infrastructure, which is exploited to recognize which plans of actions are most suited to the current situation. The system is evaluated in experiments run in a real domestic environment, showing its ability to pro-actively and smoothly adapt to subtle changes in the environment and in the habits and preferences of their user(s).

Key words: Ambient Intelligence, Smart environment, Robotic Ecology, Adaptive Planning, Self-Adaptive System, Recurrent Neural Networks.

1. INTRODUCTION

Smart environments augment our normal physical surroundings with embedded computational devices to support the delivery of services to users, anytime and anywhere. Endowing these ubiquitous devices with intelligent behaviour, and thus creating intelligent environments, is termed Ambient Intelligence (AmI).

Robotics has embraced the AmI vision by moving away from the historical view of monolithic control systems that run on specific computational units in charge of their own 'hard-wired' hardware, to promote the integration of multiple robots with larger, open distributed networks made up of different sensors, effectors, and computational units. Notable examples of such an approach include work in network robot systems (NRF, 2014; Akimoto and Hagita, 2006; Sanfeliu et al., 2008), sensor-actuator networks (Dressler, 2006), ubiquitous robotics (Kim et al., 2004), and PEIS[1] Ecologies (Saffiotti et al., 2008).

The role of robots in a smart environment is multi-faceted. From the user's perspective, the robots act as an intelligent user interface to the smart environment. Even when combined with basic Human-Robot Interaction (HRI) capabilities and simple mobile robots, such an approach can already enable a number of useful services. From the robots' perspective, being embodied in a smart environment brings a number of advantages due to the augmentation of their interaction capabilities with the sensors and actuators embedded within the smart environment, which then act as

[1] PEIS stands for Physically Embedded Intelligent Systems.

a service provider and a shared information space for both the human users and the robots inhabiting them.

Smart robotic environments made out of heterogeneous devices need intelligent decision making to decide which combination of these devices should cooperate, and how, to achieve given tasks. Having multiple alternative means to accomplish application goals when multiple courses of action are available is essential to increasing the system's robustness. However, we cannot expect application developers to have to explicitly model all of these options, dependencies and preferences. Besides adding a significant burden to the design effort, the inability of modeling other (unforeseen at design time) information in the domain ultimately impacts on the effort required to configure these systems for different and changing contexts.

In order to develop this type of smart environment, and to make them feasible in a variety of applications and environments, we need to provide them with self-adaptive properties. Rather than simply reacting to the failure of some of their components, any real world solution should be able to improve the way they operate by autonomously, pro-actively and smoothly adapting to a changing environment and to different and evolving requirements. Without self-adaptive properties, building applications combining sensors, robots and other smart things will remain unfeasible, as they would require costly pre-programming phases and continuous maintenance and supervision at each new installation, when their environment is modified or when they need to suit the changing preferences and habits of their users.

The novel solution described in this paper relies on a pervasive learning infrastructure. Sensing devices spread throughout the environment do not just provide sensed data. Rather, all of the devices in the environment harness and share dis-

tributed learning mechanisms to extend the overall capabilities of the system and to drive its continuous adaptation. Note how rather than learning to recognize complex situations or how to perform complex sensing-acting strategies, our solution is based upon a type of compositional adaptation where (possibly multiple) plans of actions are created through planning and involve the activation of pre-existing capabilities while the system learns to recognize which plans are most suited to the current situation and/or to the preferences of the user.

The remainder of this paper is organized in the following manner: Section 3 introduces our solution for self-adaptive, smart robotic environment, while Section 4 reports an experimental case study performed in a smart-home scenario to test how our solution can adapt to the characteristics of a changing environment and to the personal preferences of the user. Finally, Section 5 concludes this paper and discusses future research directions.

Preliminary ideas on the solution presented in this work have appeared in a conference paper by Bacciu et al. (2014). The content of this work consistently extends such conference paper by describing a complete architecture for realizing self-adaptive smart robotic environments, extending the original system by discussing the result of the integration between planning, learning and performance self-assessment monitoring components, and by providing a totally renewed experimental assessment considering a larger set of case studies. Furthermore, all of the key phases of our experiments are documented with supplemental video material, which is attached to this paper. We also provide the link to the datasets we used to train our learning system, which we now share with the community.

2. BACKGROUND AND RELATED WORK

The work described in this paper has been informed and advances previous work in robotics, smart environments, and self-adaptive software architectures, as briefly discussed in the following sections.

Robotics: The starting point of our work stands upon past solutions to the dynamic configuration of robotic ecologies, i.e. systems made out of heterogeneous robotic devices that cooperate in the performance of possibly complex tasks. The PEIS Ecology project (Saffiotti et al., 2008) has addressed the dynamic configuration of a robotic ecology by modelling it explicitly as a search problem that can be tackled with classical AI techniques. Such an approach relies on the existence of pre-programmed domain knowledge, in the form of specific rules that a domain expert has identified to express causal, temporal, resource and information dependencies between the different components of the ecology, and how they should be used in each situation. However, without learning abilities, such an approach suffers from a number of limitations. Specifically: (i) it is often difficult or impossible for the developer to craft the exact conditions under which a configuration would succeed; (ii) if the configuration strategy is fixed it can neither adapt to potential variations of different environments, nor take into account the experiences of the system. The present paper is about adding learning capabilities to the self-configuration of a robotic ecology.

ROBEL (for ”Robot Behavior Learning”) (Morisset and Ghallab, 2008) shares many of the motivations of the system presented here. In ROBEL, a planner is used to compose alternative plans (”skills”) to perform a given task in the context

of selecting among alternative navigation modalities. The approach presented here differs from ROBEL in several respects. Firstly, we are interested in learning the adequacy of behavior at a fine-grain: our skills are not plans, but individual perception or actuation functions. We use a planner to select and combine these functions according to their fitness into a suitable plan, and we re-plan if needed. Secondly, our system supports the execution of several tasks (what we call general-purpose). Finally, we apply our technique to the more general case of a pervasive robot system, where context information for one robot may be provided by other robots or by environmental sensors.

CogX (for "Cognitive Systems that Self-Understand and Self-Extend") (M. Hanheide et al, 2015) was a European project with the high level aim to develop a unified theory of self-understanding and self-extension. The project was implemented and demonstrated in a robot, that showed the ability to extend its own knowledge by planning learning activities and carrying them out. While CogX shared some of the motivation of the current work, its focus was on stand-alone robotic agents rather than on distributed robotic ecologies.

Smart Environments: Learning abilities have been already showcased in a number of initiatives developing adaptive smart environments and for specific learning tasks. For instance, Q-learning (Watkins, 1989) or other adaptation policies based on utility maximization have been used to learn to automate appliances, such as lights, heating, and blinds (Rashidi, 2009; Tapia et al., 2004; Gallacher et al., 2013). Other works have harnessed machine learning techniques focusing on recognising user's activities in order to provide context-aware services. These systems usually rely on static sensors placed throughout the environment (Liming et al., 2012; Alam et al.,

2012; Zhang et al., 2012; Roggen et al., 2013; De et al., 2012), but can also address more dynamic deployments, by including both static and wearable sensors (Kurz et al., 2012). Jaeger (2010) proposes a centralized hybrid learning system for activity recognition integrating a fuzzy rule-based system with probabilistic learning based on Hidden Markov Models (HMMs) and Conditional Random Fields. Roggen et al. (2013) puts forward an opportunistic approach where sensor devices self-organize to achieve activity and context recognition through a modular, though still centralized, learning system based on HMMs.

All these solutions apply a data-centric perspective, in which sensor nodes are used for data collections and feature extraction while the actual learning machinery is centralized on more capable computers. In addition, learning in these systems focuses on activity recognition (assessing the state of the user) or goal deliberation (deciding what the smart environment should do to assist its user), and often neglect the sequential nature of the sensor data produced within the ecology. We are not aware of any work that like ours defines a distributed and general-purpose learning service that can be exploited to dynamically adapt a smart robotic system to its environment and to inform it on how to best operate in a variety of tasks. In addition, we consider a learning model that has been specifically designed to deal with time-series data within the class of non-linear approaches, and which is more rarely exploited in these contexts.

Self-Adaptive Software Architectures: The type of adaptation implemented in our work is closely related to the MAPE-K (monitor, analyse, plan, execute, knowledge) loop, which is the blueprint model widely used in the engineering of self-adaptive software systems (Kephart and Chess, 2003a). In particular, as in our work,

compositional software adaptation relies on the ability for re-configuring the software by exchanging some of its algorithms or components with other (pre-existing) elements in the system to meet the current environment conditions (McKinley et al., 2004). The vision of *Autonomic Computing* pushed forward by IBM in the last decade also followed similar lines (Kephart and Chess, 2003b).

Work in this area provides useful and general principles to guide the design of self-adaptive systems, such as computational reflection. However, this mainly focuses on monitoring computational aspects, such as CPU usage and network bandwidth. In contrast, the solution presented here exploits a dynamic component framework to provide a uniform interface towards heterogeneous components, enabling their dynamic configuration and monitoring their ability to satisfy actual functional requirements.

3. APPROACH

3.1. Requirements and Key Assumptions

The inputs to the system described in this paper are high-level application goals (represented in Fig. 1) corresponding to the invocation of services that need to be provided to the user. These service goals can either be autonomously invoked or explicitly activated by the users themselves (Dragone et al., 2015). For example, through a human-machine interface the user may request the ecology to clean the kitchen after she/he had her meal, or to vacuum at least once a day every room in the home. Here we assume that the question of how application goals are first invoked is resolved (with the help of one of the methods briefly outlined above), and we focus

on describing how the subsequent behaviour of the system can be finely tuned to its environment and (changing) context and thus also improve its performance over time.

The key to tackling these issues in our approach is to rely on general purpose model descriptions of the devices in the smart environment and delegate fine-grained and smooth adaptation of the system to data-driven learning approaches.

[Figure 1 about here.]

The following sections describes the different parts of our system architecture (depicted in Fig. 1), and how they interact.

3.2. Configuration Planner

The central cognitive component at this level is a **Configuration Planner** in charge of determining and monitoring the configurations and action strategies that can be used to achieve application goals that are set for the smart environment as a whole (e.g. 'clean the apartment'). By *configuration* we mean here a specific choice of what software and hardware components are activated, and how they are interconnected: namely, which data and control channels are established among those component.

The plans generated by the configuration planner may require multiple components to exchange information and pursue possibly dependent sub-tasks (e.g., 'clean dining room', 'clean kitchen'). The plan must specify which task must be performed by which component and when. Furthermore, the decision to move a robot toward a specific location will generally require the ability to track the position of the robot

during its movements: accordingly, the plan must specify which agent (e.g., a sensor plus an interpretation routine) provides this information to the robot.

The type of configuration planning we use here was initially developed in the context of PEIS Ecologies in (Lundh et al., 2008), but similar approaches were proposed, e.g., by Tang and Parker (2005), by Vig and Adams (2006), and by S. Coradeschi et al (2013).

The configuration planner is a constraint based planner (see Rocco et al. (2013) for a more detailed technical description). It is grounded on the notion of state variable, which models elements of the domain whose state in time is represented by a symbol. State variables represent parts of the real world that are relevant for the planner's decision processes. These include the actuation and sensing capabilities of the devices in the ecology, as well as physical features in the environment. For instance, a state variable can represent the actions of a given robot, whose meaningful states might be "navigating", "grasping" and "idle". Another state variable can represent the state of a given light which can be "on", "off" or "broken". Goals are also represented through specific values of state variables. The possible evolution of state variables in time are bound by temporal constraints, e.g., stating that navigation must occur *while* the light is on, or that *after* navigation is completed the robot will be located at the kitchen. We represent temporal constraints by an extension of Allen Interval Algebra (Allen, 1984).

[Figure 2 about here.]

State variables and constraints are maintained in a constraint network. The configuration planning process manipulates this network by incrementally adding vari-

ables and constraints, until the network contains a feasible plan that connects the initial state to the goals. The resulting constraint network represents one or more temporal evolutions of the state variables that guarantee the achievement of the goals under nominal conditions. The configuration planner is itself composed of several *solvers* which all manipulate the same shared constraint network. Each solver takes into account a specific type of constraint, e.g., causal, topological, information, temporal, or resource constraints. The solvers are orchestrated using a meta-CSP approach (Mansouri and Pecora, 2016). The conceptual structure of the configuration planner is shown in Figure 2.

The configuration planner is an example of continuous planning (Dean and Wellman, 1991). The planner operates in closed loop at a cycle of about 1 Hz, constantly monitoring the execution and dynamically adapting to contingencies. At each cycle, the constraint network in the planner is updated to account for the current state of the ecology as well as goals newly posted to the planner, and the solvers are re-invoked to update the plan if needed.

In the absence of previous experience, at each iteration the planner makes its decisions according to pre-programmed preferences. However, the key to the adaptation of the system is to exploit learning solutions to (i) improve its ability to extract meaning from noisy and imprecise sensed data in order to assess the status of the environment and the application (context), and (ii) adapt its control strategies to changing and evolving situations over time, from experience rather than by relying on pre-programmed rules and static context descriptions.

To this end, our architecture adopts a learning system - whose coordination-level components are realized by a **Learning Gateway** – to process time series of data

gathered by the sensors in the ecology as well as flows of information concerning the ecology state. Its outputs are used to inform the planner about events concerning the state of the environment and of the users, and also to predict the success/fail rate in pursuing specific sub-goals and/or in using given devices and other components in the ecology. The provided predictions are then exploited by the planner to alter its planning strategies from the original hard-coded preferences to reflect the changing environmental conditions or user preferences captured by the learning system, and thus take informed decisions on which functional or action option is most appropriate or likely to succeed at any given time.

3.3. Learning System

The learning system realizes a distributed learning infrastructure, based on the RUBICON Learning Layer by Bacciu et al. (2012), capable of addressing a large variety of computational learning tasks concerning the on-line processing of streams of sensor-data and system information to provide predictions regarding the ecology state (e.g. event recognition). Specifically, it proposes a neural-motivated architecture, where state-of-the-art recurrent neural networks (RNN) (Kolen and Kremer, 2001) are exploited as a computationally efficient means for implementing learning functions on-board the units composing the ecology.

The learning system predictions are generated by a distributed neural computation between the learning models deployed on the ecology devices and that are allowed to interact through a synaptic communication mechanism abstracting from the details of the underlying network topology and transmission media. The RUBICON Learning Layer (Bacciu et al., 2012) provides mechanisms that allow con-

tinuous adaptation of the learned knowledge (i.e. the predictions) by incrementally adding new learning tasks or by re-training existing ones based on the ecology needs, by interacting with the components implementing the *Performance Monitor* functions within the system. Moreover, it provides self-configuration and self-adaptation mechanisms that allow the identification of relevant information sources for the learning tasks, the automatic over-the-air deployment of learning modules on the distributed ecology devices as well as to catering for nodes dynamically joining and leaving the ecology.

Fig. 3 provides the high-level architecture of the learning system, highlighting both the distributed learning components as well as the coordination and service level entities of the system, responsible of the interaction with the *Performance Monitor* and *Configuration Planner* components. At a first glance, the system comprises a number of coordination level subsystems, running on the Learning Gateway, which implement the learning system management and coordination functions, together with bulkier mechanisms associated with the incremental acquisition of new learning tasks. At this level, the Learning Gateway also realizes the interfacing with other coordination level instances in the ecology. The Learning Network subsystem, on the other hand, is a collection of software components implementing the learning modules on the distributed nodes of the ecology and it ultimately computes the the run-time predictions of the system. Such learning modules are the key service-level machinery of the RUBICON Learning Layer and have been designed based on two cornerstones, namely

- the need to deal with temporal noisy data, typically observed in ubiquitous robotic scenarios, and
- the heterogeneity of the computational resources of the ecology devices, which demands learning models capable of limited computational requirements.

[Figure 3 about here.]

Based on such considerations, the learning modules have been realized by means of Leaky Integrator Echo State Networks (LI-ESNs) (Jaeger and Haas, 2004; Jaeger et al., 2007), a recurrent neural model from the Reservoir Computing (RC) (Lukosevicius and Jaeger, 2009) framework, characterized by a good trade-off between computational efficiency and ability to deal with noisy time-series data (Bacciu et al., 2014). LI-ESNs are used to model discrete-time input-driven dynamical systems and are composed of a non-linear recurrent part, called reservoir, which implements the non-linear dynamical memory of the system through a state representation, and of a (typically) linear feed-forward component, called readout, which exploits the richness of the reservoir dynamics to linearly compute the output.

[Figure 4 about here.]

A graphical illustration of the LI-ESN architecture is provided in Fig. 4. Specifically, in our implementation, at each time step t the LI-ESN module receives in input a vector containing the readings from a set sensors from the devices of the robotic ecology, i.e. $\mathbf{s}(t) = [s_1(t), s_2(t), \ldots, s_{N_{in}}(t)]^T \in \mathbb{R}^{N_{in}}$, where each $s_i(t)$ denotes a sensed information at time t and N_{in} is the total number of considered inputs. Denoting the reservoir size (the number of reservoir's neurons) by N_{res}, the state of the LI-ESN at time t, i.e. $\mathbf{x}_{res}(t) \in \mathbb{R}^{N_{res}}$, is computed as a function of the input at

time t and of network's state at time $t-1$, according to the following state transition equation:

$$\mathbf{x}_{res}(t) = (1-a)\mathbf{x}_{res}(t-1) + af_{res}(\mathbf{W}_{in}\mathbf{s}(t) + \mathbf{W}_{res}\mathbf{x}_{res}(t-1)), \quad (1)$$

where $a \in [0,1]$ is the leaky parameter that governs the speed of the state dynamics in reaction to the input, f_{res} denotes the element-wise applied activation function of the reservoir units (we use $tanh$), $\mathbf{W}_{in}$ is the input-to-reservoir weight matrix and $\mathbf{W}_{res}$ is the recurrent reservoir weight matrix.

At each time step t the readout computes the output of the LI-ESN network by means of a linear combination of the elements in the state $\mathbf{x}_{res}(t)$. In our application, the LI-ESN modules are used to predict a preference weight value for the planner, thereby, denoting by $y_w(t)$ the prediction computed as output by the LI-ESN at time t, we have:

$$y_w(t) = \mathbf{w}_{out}\mathbf{x}_{res}(t), \quad (2)$$

where $\mathbf{w}_{out}$ is the reservoir-to-readout weight vector.

The extreme efficiency of LI-ESN training, with respect to standard RNN, stems from the fact that only the readout component requires a training process. In particular, the elements in $\mathbf{w}_{out}$ in eq. 2, which represent the parameters of the readout, are adjusted on a training set, typically by means of direct methods such as ridge regression with regularization parameter λ_r. The parameters of the reservoir component, i.e. the values in $\mathbf{W}_{in}$ and $\mathbf{W}_{res}$ in eq. 1, are set according to the Echo State Property (see Jaeger (2001); Yildiz et al. (2012)), by scaling the spectral radius of its recurrent weight matrix (Lukosevicius and Jaeger, 2009) to a value that is smaller than 1 (we used the typical value of 0.9). The key hyper-parameters of LI-

ESN networks considered here are: (i) the reservoir size N_{res} (i.e. the dimension of the state space), (ii) the leaky parameter a and (iii) the readout regularization λ_r, used for ridge regression. Further information on the characteristics of the RC approach can be found in the literature (Jaeger and Haas, 2004; Lukosevicius and Jaeger, 2009; Gallicchio and Micheli, 2011).

The Learning Network realizes an adaptive environmental memory that embeds knowledge within the LI-ESN learning modules distributed in the environment. At run-time, the single LI-ESN processes device-local information, e.g. gathered by the on-board sensors of a mote, and integrates this with remote inputs received from other learning modules and delivered through the synaptic communication mechanism Bacciu et al. (2012). The learning module implementation (and associated synaptic communication) is available as a NesC library Gay et al. (2003) for low-power TinyOS-enabled WSN motes, and as a Java API for more powerful devices.

The predictions generated by the Learning Network through the distributed neural computation are collected by the coordination level component referred to as LN (Learning Network) Interface in Fig. 3. This component provides an entry point to the learning network subsystem, providing methods to interact with the learning modules and allowing abstraction from implementation and deployment details, such as distribution. This interface also takes care of publishing the learning system predictions to the other coordination-level components of the system, such as the Configuration Planner.

The learning system provides a variety of mechanisms for continuous adaptation of the knowledge captured by the Learning Network. The Training Agent component (deployed at coordination level on the Learning Gateway) manages the

learning phases of the system, handling the training information received from the *Performance Monitor* instances of the ecology. In particular, it implements the more computationally demanding processes of the system that include mechanisms for:

- *Incremental Learning*, allowing dynamic acquisition of new learning tasks at run-time or retraining from scratch the existing ones by exploiting a mirrored copy of the deployed learning models maintained by the Network Mirror component in Fig. 3;
- *Unsupervised Feature Selection*, based on the Iterative Cross-correlation Filter algorithm by Bacciu (2016), that allows automatic identification of those input sources that are either redundant or provide irrelevant information;
- a model selection mechanism with integrated *Supervised Feature Selection* for RC (Bacciu et al., 2015), allowing automatic selection of the best learning model configuration and parameters and the determination of the most-predictive input sources for a computational learning task.

These three mechanisms are integrated to provide the learning system with self-configuration and self-adaptation capabilities. In particular, the end-to-end process of training and deploying a new learning model can be summarized as follows:

(1) First, the feature selection mechanisms are used as a preliminary step of the incremental learning chain to identify a compact set of predictive and non-redundant input sources for the new task from the initial candidate superset of sources provided by the *Performance Monitor*.
(2) The Training Agent exploits the model selection mechanism to identify the best-

performing configuration of the learning model that is trained using only the input information identified by the feature selection process.

(3) The incremental learning functions are used to allocate the newly-trained learning model on the most adequate device of the ecology, using an allocation policy that favours minimizing the communication effort required to bring all the learning model inputs to the hosting device.

(4) Finally, the Training Agent deploys the newly trained learning module on the target device using over-the-air communication and appropriately sets up the associated synaptic communication (e.g. to feed the learning module inputs and to transfer its predictions).

3.4. Performance Monitor

The final piece of the system in Fig. 1 is the **Performance Monitor**, that mediates the interaction between the Configuration Planner and the Learning Gateway. Specifically, the responsibility of the monitor is to collect and analyse the performance of the ecology and its components, and to trigger the re-training of the learning system whenever this fails to be satisfactory or deteriorates in a way to suggest that the planner needs to adapt its configuration and/or action strategy. To this end, the monitor maintains a set of *performance maps*, each corresponding to an aspect of interest for the ecology. For instance, a *navigation performance map* is maintained to collect information on the performance of the navigation goals executed within the system. For the purpose of this work's applications, the granularity of the map can be limited to the rooms composing the indoor environment where the system is deployed, i.e. the navigation map is topological and maintains

information on the performance of all navigation tasks ending in each room of the apartment. Different maps can be maintained for different components (e.g., a different localization systems) in order to monitor their performance at a finer granularity.

Our monitor component is implemented using the *Self*-OSGi framework (Dragone et al., 2012), a Java based tool designed to supervise component-based systems. Specifically, we have used *Self*-OSGi to implement monitoring components (managers) to monitor the most important sub-systems of the ROS navigation stack. These include:

- **NavigationManager**: this monitor supervises the instantiation of any sub-systems necessary to perform map-based navigation. It also exports a PEIS interface to accept navigation goals that are forwarded to the underlying ROS system in order to instruct the robot to move to given positions. Once a navigation goal is received and forwarded to the navigation stack, the monitor subscribes to the ROS action feedback reporting the estimated progress of the robot on its way to its current target, and publishes the feedback in PEIS, for the benefit of high-level controllers.
- **LocalisationMonitor** this monitor supervises localisation functions. Available implementations include the standard particle filter included (implemented with localisation sub-systems included in the standard ROS navigations stack, which uses a particle filter to track the pose of a robot against a known map, or a localisation component exploiting an RFID reader mounted on the robot navigating over an RFID-equipped floor). In addition, it estimates the quality of the localization

provided by these sub-systems, and publishes the quality estimate on the PEIS tuplespace.

Finally, our monitoring system includes a **Logging** service to configure, start and stop the logging of raw-sensor data but also higher-level information, such as the robots' estimated locations. The resulting logs are recorded as sequences of time-stamped tuples into a relational database.

4. EXPERIMENTS

4.1. Experimentation Goal

We tested the approach outlined in the previous section in a practical application comprising a home environment where a simple mobile robot operates in support of a user. The goal of the experiments is to show the capabilities of the system in adapting to both changes in the characteristics of the environment and to user personal preferences. In particular, we assess the ability of the system in identifying performance issues and dynamically deploying a learning task to address them, autonomously selecting relevant information sources and handling learning module training and deployment.

Two experimental case studies have been designed to exercise system self-adaptation[2]. The first task, referred to as Entrance Mirror, concerns an ambient alteration inducing disturbances to the laser-based localization system of the mobile robot when navigating in certain areas of the flat. The second task, referred to as Kitchen Cleaning,

[2]Datasets can be downloaded here: http://pages.di.unipi.it/bacciu/angendatasets/(files will be uploaded to a public repository before paper publication).

concerns the adaptation to a user's preference of not sharing the kitchen with a cleaning robot. In both case studies, the system is confronted with a recurrent failure or performance degradation when pursuing an ecology goal (e.g. navigation or house cleaning), that is induced by un-modelled aspects of the tasks. We show how the self-adaptation abilities of the system allow it to change its execution strategies to avoid those action plans that produce failure or performance degradation.

4.2. Experimental Setup

The experimental scenario has been realized in the Ängen research and innovation apartment,[3] which is part of the Ängen senior residence facility in the city of Örebro, Sweden. It comprises a real-world flat sensorized by an RFID floor, a Turtlebot (www.turtlebot.com) mobile robot with range-finder localization and a Wireless Sensor Network (WSN) with six mote-class devices from the TelosB family (Crossbow, 2011). The robot performs actions in support to the user's daily activities which entail navigating between different rooms, as depicted in Fig. 5(a). The WSN motes are distributed in the kitchen and across the living room and entrance, as detailed in Fig.5(b) where the term M_i is used to denote the i-th mote. Each device monitors the environment via light (L), temperature (T), humidity (H) transducers and passive infrared (P) presence sensors, for a total of 24 sensor streams being collected at a 2Hz rate. The snapshots of the mote locations in Fig. 5(b) show that some devices are in a position allowing the presence sensors to be triggered by both the robot and user motion, e.g. M_3 and M_6, while others are triggered solely by user presence, e.g. M_2.

[3] http://angeninnovation.se

[Figure 5 about here.]

The system implemented in the testbed deploys the following software components: (i) a version of the TinyOS Learning Network API (implementing synaptic communication and learning functions for the WSN motes), and (ii) an embedded WSN control component for the purpose of node management and for training data collection. A gateway device, connected to the WSN sink through a serial interface, has been deployed to run instances of the Learning Gateway API as well as of components of the Control Layer including the whole agent monitoring suite described in the previous section. Communication between each monitor components and the localization and navigation sub-systems running on the robot rely on the robotic operating system, by exploiting the integration between the agent system and ROS, implemented over ROSJava (Kohler and Conley, 2011).

The experiments explore how the learning system responds when provided with different initial sets of information sources. To this end, we consider a set of candidate inputs that include all information sources available, i.e. the 24 sensor streams from the motes and the robot position and pose (x, y coordinates and orientation θ). Figure 6 shows a snapshot of the data for the Entrance Mirror experiment which clearly highlights the considerable amounts of irrelevant/noisy input features out of which the system has to automatically identify relevant predictors for the preference weight. In this respect, we also assess the contribution of the feature selection mechanisms made available by the learning system (see Section 3.3), by confronting the predictive performance when using all the 27 available features (*global* configuration) with respect to using the attribute identified by feature selection (*global feature*

selection configuration). In addition, we consider two configurations, referred to as *local* and *global ad-hoc*, using expert knowledge concerning the most relevant input sources for the two tasks. The former uses only selected information from a single mote whose transducers capture the most relevant (i.e. predictive) information for the task: e.g., mote M_3 in Fig. 5(b), for the Kitchen Cleaning task, as its presence sensor is triggered by both the robot and the user entering the kitchen. The global ad-hoc configuration integrates selected information from multiple (task-relevant) sensors. Each input configuration has undergone an automated hold-out model selection procedure exploring RC hyper-parameters to seek the best module configuration. Specifically, the system is configured to train and validate LI-ESN modules with reservoir size N_{res} in $\{10, 50, 100, 300, 500\}$, leaky parameter a in $\{0.1, 0.5, 1\}$ and readout regularization λ_r in $\{1000, 100, 10, 1, 0, 0.1, 0.01, 0.001\}$. Test sets for the experiments have been generated by holding out $\approx 30\%$ of the data collected by the Performance Monitor component, while the remaining $\approx 70\%$ serves for training and validation purposes.

[Figure 6 about here.]

4.3. Entrance Mirror Experiment

4.3.1. *Goal and Setup.* The goal of the first experiment is to test the ability of our system to autonomously adapt to modifications to the environment affecting the performance of one of the localization sub-systems available to the robot. In particular, we consider how an environment characteristics unforeseen at design time can introduce a degradation in the ecology ability in pursuing navigation goals. Then, we show how the adaptation mechanisms can be used to revise the background

knowledge imbued in the planner before deployment to cope with the unforeseen environment aspect, hence avoiding recurrent failures and performance degradation.

We consider a robot that can track its position either by exploiting the standard Adaptive Monte Carlo Localization (AMCL) algorithm included in the ROS navigation stack by using data captured by its on-board RGBD Kinect camera ?? (kin), or by using an RFID reader to detect the RFID tags placed under the floor of the apartment. While the second localization system is less expensive, in terms of energy consumption and of resource computation needed, it is also less precise. For this reason, the system is configured with factory preference weights that favor the use of AMCL localization in default conditions (e.g. in absence of device faults).

The experiment concerns the introduction of an ambient alteration obtained by positioning a large mirror close to the apartment entrance, depicted as a rectangle on the map in Fig. 5(a). The presence of the mirror inhibits the effectiveness of the Kinect device when the robot navigates along certain paths, as the structured infra-red light emitted by that sensor is reflected by the mirror (see the snapshot in Fig.7). In particular, the Kinect is affected in certain curved trajectories facing the mirror (dotted red arrow in Fig. 5(a)), while other straight trajectories will generally not be affected by the the mirror (continuous green arrow in Fig. 5(a)). When the robot moves along the curved trajectories, the AMCL algorithm is confused since the corridor suddenly appears to have a large opening where none was previously recorded in the map of the environment (see Fig. 8 for an example of perturbed localisation).

[Figure 7 about here.]

[Figure 8 about here.]

As a result of this ambient alteration, we expect a quality degradation of the AMCL pose when performing navigation tasks involving trajectories facing the mirror. Despite such performance degradation, a non-adaptive system would keep using the AMCL localization coherently with the preference weight encoded at factory time which, however, is no longer consistent with the new ambient conditions. On the other hand, we expect an adaptive system to identify, over time, such degradation and to learn to pro-actively downgrade preference towards AMCL when the distorting conditions occur. Figure 9 shows the fixed preference weights of a non-adaptive system for RFID and AMCL localization compared with predicted preference weights for AMCL under two conditions, one in which a the localization system will be likely to incur in a disturbance and one without. One can clearly see how using a predictive approach, would allow steering of the preference towards RFID-based localization only in those situations in which this is actually more robust than AMCL localization.

[Figure 9 about here.]

4.3.2. *Experiment Description.* We have instructed the robot with a series of tasks causing it to move along both curved and straight trajectories, simulating robot daily activities to accelerate data collection over a short period of time. The monitoring system described in the previous section has been used to collect evidence of the performance of the robot's localization system on the single trajectories. The quality of the actual trajectory performed by the robot using the localization system has been assessed in terms of oscillation of the magnitude of the instantaneous robot

speed with respect to the average speed on the trajectory. The underlying intuition is that a well functioning service robot in a home environment will move smoothly without considerable speed jumps. More formally, the navigation performance w_t at time t is

$$w_t = \exp\left(-(\overline{v}_t - \mu)^2\right) \tag{3}$$

where μ is the average magnitude of the speed on the full trajectory. The term $\overline{v}_t$ is recursively defined as

$$\overline{v}_t = \max\{\overline{v}_{t-1}, v_t\} \tag{4}$$

where v_t is the magnitude of velocity at time t and Eq. (4) ensures a monotonically decreasing performance measure in (3).

The monitoring component records all relevant status information during task execution (i.e all WSN sensors and robot positioning data), it assesses the quality of navigation using (4) and updates the aggregated statistics for the performance map associated with the navigation task. For this experiment, it has autonomously collected 65 example trajectories (with data sampled at 2Hz). Figure 10 plots the w_t value for the single trajectories as a function of time, each trajectory being represented by a different line: note how mirror influence results in the majority of the curved trajectories having target values ending under 0.5, that is the threshold for deeming navigation performance satisfactory.

[Figure 10 about here.]

The example data collected by the monitoring component is fed to the distributed learning system so that it can learn to predict the quality of AMCL navigation under

the effect of the unmodeled disturbance (i.e. the mirror). The distributed learning system is thus supplied with the 65 sequences of 24 transducers readings and 3 measurements concerning robot location and pose values, as well as the corresponding target values in Fig. 7. Upon reception of such data, the Learning Layer autonomously deploys a new learning task by

(1) identifying what input information is worth using from the superset provided by the
(2) training and validating the learning module;
(3) deploying the final preference weight predictor as well as the supporting synaptic

To provide a baseline performance for feature selection and to assess the effect of distribution we have also tested alternative configurations of the input data, as described in Section 4.2.

4.3.3. *Results.* Table 1 shows the detail of the learning model inputs for the various configurations: the *global feature selection* configuration reports the outcome of the feature selection process. Note how this mechanism is able to automatically identify a compact subset of input sources (i.e. 6) from the initial set of 27 features provided by the supervisor component. This subset is coherent with the ground truth configuration *global ad-hoc* that is based on expert knowledge on the input sources actually relevant for the task (e.g. based on mote location in Fig. 5), showing the self-configuration capabilities of the system as it pertains to the autonomous identification of data sources for dynamically acquired learning tasks.

[Table 1 about here.]

Each input configuration has been assessed through the automated model selection procedure available in the system and exploring the RC hyper-parameters values in Section 4.2 to seek the best learning module configuration. Figure 11 shows the Mean Absolute test Error (MAE) achieved by the LI-ESN as a function of the reservoir size (averaged on 10 reservoir guesses). The beneficial effect of feature selection is evident, since considering all available streams of information (i.e. the global configuration) inevitably leads to poorer performances as the size of the reservoir increases. This is due to the fact that larger learning models are more prone to fit irrelevant and noisy information arising from non-significant inputs for the task, e.g. light and humidity readings, resulting in a reduction of the quality of the learned predictor. On the other hand, the input configuration identified by the feature filtering phase yields comparable results to input configurations based on ad-hoc expert knowledge. Note how, for this particular task, using only information local to a single, but very relevant for the task, WSN device is sufficient to achieve a performance that is basically equivalent to that of the global ad-hoc configuration.

[Figure 11 about here.]

A deeper look at the ground truth weights in Fig. 10 highlights the noisy nature of the target function, where some straight trajectories which are not influenced by the mirror show, nevertheless, poor navigation performances. On the other hand, a few curved trajectories do not seem to be influenced by mirror presence, showing target values stably over 0.5. This defines a very challenging learning task where the appropriate satisfaction weight is not readily predicted. Despite such conflicting target information, the learning system achieves a good predictive performance that enables

the control components to take more informed choices concerning which navigation system to use. In particular, if considering the predicted preference weights (in place of the hard-coded AMCL weight) to determine which navigation system to use, the control system would have increased the number of Entrance navigation tasks completed successfully by a 20%, effectively avoiding the disruption introduced by the unmodeled mirror presence on specific trajectories and which is documented on the video[4] snapshots in Fig. 12.

[Figure 12 about here.]

4.4. Kitchen Cleaning Experiment

4.4.1. *Goal and Setup.* The second experiment tests the ability of our system to autonomously adapt to un-modeled user preferences. We have considered a scenario where a user prefers not to have the cleaning robot (simulated by a Turtlebot platform) operating in the kitchen while he is in the same room (the same approach can be generalized to further rooms in the flat). Without the ability to identify such preferences and adapt the plans of the ecology accordingly, the system will keep sending the robot to clean the kitchen also when the user is present, resulting in the user being bothered by the system and in a number of failures of the cleaning task due to the abortion forced by the user. We show how the system is capable of identifying such a user preference from the aborted cleaning tasks and to learn an associated preference weight that provides the planner with a context-aware indication of when to avoid sending the robot to clean the room.

[4]Videos can be downloaded here: `http://pages.di.unipi.it/bacciu/entrance_task/` (files will be uploaded to a public repository before paper publication).

4.4.2. *Experiment Description.* We have instructed the robot with a series of cleaning tasks in the kitchen to accelerate the time-span of daily data collection. For the sake of simplicity, it has been assumed that the robot was always located in the living room when the kitchen cleaning goal was posted, but the approach can be generalized to have the origin in any room of the flat. The stereotypical navigation performed by the robot is shown as a dashed blue arrow in Fig. 5(a). We have run a total of 50 tests in which we have also instructed a human volunteer to simulate daily presence in the flat, by requiring him to

(1) be somewhere outside of the kitchen for the entire length of the test, so that the robot was left undisturbed to move to the kitchen and clean it;
(2) be and remain in the kitchen for the whole test;
(3) be in the kitchen at the beginning of the test but leave and move to another room afterward, before or just after the robot entered in the kitchen;
(4) be somewhere else outside the kitchen (e.g. hall, bedroom, bathroom) and walk into the kitchen at some point of the test.

The monitoring system has been used as in the previous section to collect information from all the flat sensors and the robot's pose and orientation. Performance assessment, in this task, is associated with the successful completion of the cleaning duties by the robot and can hence be related with goal achievement.

We assume that the robot can complete its duties if the user is not in the kitchen, whereas, when the user is in the kitchen it prevents the robot from completing its goal by some form of user-system interaction mechanism (e.g a simple one would use a robot bumper to provide a negative feedback for the action). For the sake

of this experiment, user feedback has been added at post processing, by adding a negative feedback (i.e. a 0 target preference) whenever the robot attempted to clean the kitchen while the user was already in the room or whenever he entered the kitchen while the robot was cleaning. We provided a positive feedback (i.e. a target preference equal to 1) in all the other cases.

4.4.3. *Results.* The learning system has been tested using the input configurations summarized in Table 2: again, the *global feature selection* configuration reports the outcome of the feature selection process on all the 27 available inputs. It can be noted how this mechanism enforces the self-configuration capabilities of the distributed learning system by reducing the number on final inputs to 6 very significant information sources. Such an ability is essential in a resource constrained scenario, where fewer inputs means less information to be transferred over-the-air and less complexity in the final learning model (thus less costly memory fingerprints). In particular, based on the sensors' location in Fig. 5, it is clear that the PIR sensors detected as relevant by the feature selection mechanisms are all those that are triggered by either user or robot movements in the flat. Notably, the feature selection mechanism also filters out the y-position of the robot and its orientation, which is coherent with the fact that the robot moves, essentially, on horizontal trajectories along the x-axis.

[Table 2 about here.]

The test MAE of the models selected by the validation procedure is shown in Fig. 13: the positive effect of considering only relevant input information to predict the preference weight is quite neat, with the configuration using all available infor-

mation experiencing a clear performance decrease (i.e. an increase in the test error) as the size of the reservoir increases. Conversely, the feature selection configuration achieves an identical performance to the ad-hoc configuration exploiting expert knowledge: see also the results for the model-selected configurations in Table 3. The integration of information captured by different WSN motes is essential to ensure a good prediction quality, with the ad-hoc configuration using information from a single mote considerably under-achieving with respect to a distributed approach (see the low performance values of the local configuration in Table 3).

[Table 3 about here.]

Finally, the results in Fig. 13 also suggest that a learning module with a small reservoir of only 50 units already presents a good predictive performance. This aspect is key for actual learning model deployment, as a LI-ESN with a 50-unit reservoir has a memory fingerprint that makes it suitable for being embedded also in the ecology nodes with lowest computational capabilities, such as the WSN motes. In this respect, and also to assess the full self-adaptation pipeline of our system, we have tested the actual deployment of the 50-unit learning module resulting from the feature selection configuration. Figure 14 shows snapshots of the video demo[5] documenting how the ESN predictions are used by the control system to dynamically adapt the scheduling of the mobile robot task to comply with the user preferences and to its presence pattern in the kitchen. For instance, it shows the Configuration Planner having learned to dynamically avoid or defer the kitchen cleaning task when

[5]Demo video can be downloaded here: `http://pages.di.unipi.it/bacciu/kitchen_task/`(files will be uploaded to a public repository before paper publication).

the user is stably detected in the kitchen by the presence sensors, e.g. rescheduling the robot activities to execute an alternative navigation in the meanwhile.

[Figure 13 about here.]

[Figure 14 about here.]

5. CONCLUSION AND FUTURE WORK

We have presented an Ambient Intelligence approach for self-adaptive robotic ecologies and described a series of experiments that we have carried out to evaluate its effectiveness in real application settings. Our solution is able to monitor the performances and the outcomes of its own actions and to exploit a pervasive learning service to learn that there are (previously unknown) dependencies between certain sensor inputs and the suitability of certain plans of actions and system configurations out of a set of suitable options to achieve its goals.

One of the key aspect of the proposed approach is the fact that it provides an effective means to integrate a-priori available knowledge concerning the application, the environment and the systems, encoded in the rules and constraints of the configuration planner, with the adaptation capabilities of a sub-symbolic memory, implemented by the distributed learning system. In this respect, the distributed learning system acts as an interpreter and bridge between the noisy contextual information gathered through the ambient sensors, and the higher order functions of the ecology in the planner, by relaying on the simple concept of learning adaptive action and plan preference weights. Our system also allows embedding in the environment the contextual knowledge acquired by the ecology, by deploying the components of the

learning system on the distributed nodes of the ecology, even on very low power ones such as tiny mote devices within a 802.15.4 wireless sensor network.

We have described an instantiation of our system on a real-world ambient intelligence scenario. Our solution uses a number of performance-monitoring components and integrates them with a configuration planner and with an agent-based logging and monitoring framework. The resulting system can autonomously provide teaching signals to the learning service, and exploit its predictions to smoothly and autonomously adapt to user's preferences and changing environments. This is a very important improvement over the use of hard-coded strategies. Compared to existing works and specific examples of robot's learning and learning for smart environments, our approach represents a novel solution for (i) the way it can be applied as a general purpose learning service, and (ii) for its seamless exploitation of both robots' onboard and ambient sensors. In terms of existing work, our approach also explicitly tackles the timeseries nature of the data that is typically involved in the ambient sensing, by proposing an approach founding on recurrent neural networks in place of typical approaches in literature which rely on learning models for static, flat data. Our results show that these ESN neural models can be used to provide efficient and modular recurrent neural network models tailored to the very low-computational capacity of the cheap sensor mote used in robotic ecologies.

Overall, these advancements can greatly simplify design, customization and adaptation of smart robotic environments, and ultimately enhance the user acceptance of this technology with systems constantly tailored to users' needs. What makes our solution a practical one, which limits the computational cost of our learning methods and enables us to use fully automatic feature selection algorithms, is (i) the existence

of a finite set of pre-defined goals and control options, and (ii) the reduced number of sensor sources included in current smart homes.

Future research should increase the scalability of these systems, and address the challenging problem of coupling them with general purpose cognitive mechanisms to drive efficient system's exploration of control options. Further, we would like to investigate the advantages of having a distributed learning system. In this respect, we foresee two potentially impacting contributions: on the one hand, we expect the ability to process and fuse information locally, close to where such data is generated (e.g. directly on-board the wireless sensor motes), to allow reducing communication effort and thus power consumption. On the other hand, since the system distributes the knowledge acquired by the learning system across the environment, we would like to assess how distributions impacts sharing of such learned knowledge between agents (robots, actuators).

REFERENCES

Microsoft kinect for xbox. `http://www.xbox.com/en-GB/kinect/`.

AKIMOTO, T., and N. HAGITA. 2006. Introduction to a Network Robot System. *In* Intelligent Signal Processing and Communications, 2006. ISPACS '06. International Symposium on, pp. 91 –94.

ALAM, M.R., M.B.I. REAZ, and M.A. MOHD ALI. 2012. Speed: An inhabitant activity prediction algorithm for smart homes. Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, **42**(4):985–990. ISSN 1083-4427. .

ALLEN, JAMES F. 1984. Towards a general theory of action and time. Artif. Intell., **23**(2):123–154. ISSN 0004-3702. .

BACCIU, DAVIDE. 2016. Unsupervised feature selection for sensor time-series in pervasive computing applications. Neural Computing and Applications, **27**(5):1077–1091.

BACCIU, D., P. BARSOCCHI, S. CHESSA, C. GALLICCHIO, and A. MICHELI. 2014. An experimental characterization of reservoir computing in ambient assisted living applications. Neural Computing and Applications, **24 (6)**:1451–1464. ISSN 0941-0643.

BACCIU, DAVIDE, FILIPPO BENEDETTI, and ALESSIO MICHELI. 2015. Esnigma: efficient feature selection for echo state networks. *In* Proceedings of the European symposium on artificial neural networks, computational intelligence and machine learning (ESANN15), pp. 189–194.

BACCIU, D., S. CHESSA, C. GALLICCHIO, A. LENZI, A. MICHELI, and S. PELAGATTI. 2012. A general purpose distributed learning model for robotic ecologies. *In* Robot Control, Volume 10, pp. 435–440.

BACCIU, D., M. DI ROCCO, C. GALLICCHIO, A. MICHELI, and A. SAFFIOTTI. 2014. Learning context-aware mobile robot navigation in home environments. *In* Proc. of the 5th Int. Conf. on Information, Intelligence, Systems and Applications (IISA 2014), IEEE, pp. 57–62.

CROSSBOW. 2011. Datasheet of telosb.

DE, DEBRAJ, SHAOJIE TANG, WEN-ZHAN SONG, DIANE COOK, and SAJAL K. DAS. 2012. Activity-aware sensor network in smart environments. Pervasive and Mobile Computing, **8**(5):730 – 750. ISSN 1574-1192.

DEAN, T.L., and M.P. WELLMAN. 1991. Planning and control. Morgan Kaufmann Publishers Inc.

DRAGONE, M., S. ABDEL-NABY, D. SWORDS, G.M.P. O'HARE, and M. BROXVALL. 2012. A programming framework for multi-agent coordination of robotic ecologies. *In* ProMAS 2012, pp. 72–89.

DRAGONE, MAURO, GIUSEPPE AMATO, DAVIDE BACCIU, STEFANO CHESSA, SONYA A. COLEMAN, MAURIZIO DI ROCCO, CLAUDIO GALLICCHIO, CLAUDIO GENNARO, HÉCTOR LOZANO PEITEADO, LIAM P. MAGUIRE, T. MARTIN MCGINNITY, ALESSIO MICHELI, GREGORY M. P. O'HARE, ARANTXA RENTERÍA, ALESSANDRO SAFFIOTTI, CLAUDIO VAIRO, and PHILIP J. VANCE. 2015. A cognitive robotic ecology approach to self-configuring and evolving AAL systems. Eng. Appl. of AI, **45**:269–280.

DRESSLER, F. 2006. Self-organization in autonomous sensor/actuator networks. *In* In Proc. of the Int Conf on Architecture of Computing Systems.

GALLACHER, S.M., E. PAPADOPOULOU, N.K. TAYLOR, and M.H. WILLIAMS. 2013. Learning user preferences for adaptive pervasive environments: An incremental and temporal approach. ACM Transactions on Autonomous and Adaptive Systems, **8**(5).

GALLICCHIO, C., and A. MICHELI. 2011. Architectural and markovian factors of echo state networks. Neural Networks, **24**(5):440–456.

GAY, DAVID, PHILIP LEVIS, ROBERT VON BEHREN, MATT WELSH, ERIC BREWER, and DAVID CULLER. 2003. The nesc language: A holistic approach to networked embedded systems. *In* Proceedings of the

ACM SIGPLAN 2003 conference on Programming language design and implementation, PLDI '03, ACM, pp. 1–11.

JAEGER, H. 2001. The "echo state" approach to analysing and training recurrent neural networks. Technical report, GMD - German National Research Institute for Computer Science.

JAEGER, H. 2010. Ksera project: Deliverable d4.1 learning & decision making algorithms in pervasive environments. Technical report.

JAEGER, H., and H. HAAS. 2004. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wirelesscommunication. Science, **304**(5667):78–80.

JAEGER, H., M. LUKOŠEVIČIUS, D. POPOVICI, and U. SIEWERT. 2007. Optimization and applications of echo state networks with leaky-integrator neurons. Neural Networks, **20**(3):335–352.

KEPHART, J.O., and D.M. CHESS. 2003a. The vision of autonomic computing. Computer, **36**(1):41–50. ISSN 0018-9162. .

KEPHART, JEFFREY O, and DAVID M CHESS. 2003b. The vision of autonomic computing. Computer, **36**(1):41–50.

KIM, JONG-HWAN, YONG-DUK KIM, and KANG-HEE LEE. 2004. The Third Generation of Robotics: Ubiquitous Robot. *In* Proc of the 2nd Int Conf on Autonomous Robots and Agents.

KOHLER, DAMON, and KEN CONLEY. 2011. rosjava–an implementation of ros in pure java with android support.

KOLEN, J.F., and S.C. KREMER editors. 2001. A Field Guide to Dynamical Recurrent Networks. IEEE Press.

KURZ, MARK, HLZL GEROLD, ALOIS FERSCHA, ALBERTO CALATRONI, DANIEL ROGGEN, GERHARD TRSTER, HESAM SAGHA, RICARDO CHAVARRIAGA, JOS DEL R. MILLN, DAVID BANNACH, KAI KUNZE, and PAUL LUKOWICZ. 2012. The OPPORTUNITY Framework and Data Processing Ecosystem for Opportunistic Activity and Context Recognition. International Journal of Sensors, Wireless Communications and Control, **1**(2):102–125.

LIMING, C., C.D. NUGENT, and W. HUI. 2012. A knowledge-driven approach to activity recognition in smart homes. Knowledge and Data Engineering, IEEE Transactions on, **24**(6):961–974. ISSN 1041-4347. .

LUKOSEVICIUS, M., and H. JAEGER. 2009. Reservoir computing approaches to recurrent neural network training. Computer Science Review, **3**(3):127 – 149. ISSN 1574-0137.

LUNDH, ROBERT, LARS KARLSSON, and ALESSANDRO SAFFIOTTI. 2008. Autonomous functional configuration of a network robot system. Robot. Auton. Syst., **56**(10):819–830. ISSN 0921-8890. .

M. HANHEIDE ET AL. 2015. Robot task planning and explanation in open and uncertain worlds. *In* Artificial Intelligence. In press. DOI: http://dx.doi.org/10.1016/j.artint.2015.08.008.

MANSOURI, MASOUMEH, and FEDERICO PECORA. 2016. A robot sets a table: a case for hybrid reasoning with different types of knowledge. Journal of Experimental & Theoretical Artificial Intelligence, **28**(5):801–821.

MCKINLEY, PHILIP K., SEYED MASOUD SADJADI, ERIC P. KASTEN, and BETTY H. C. CHENG. 2004. Composing adaptive software. Computer, **37**(7):56–64.

MORISSET, BENOIT, and MALIK GHALLAB. 2008. Learning how to combine sensory-motor functions into a robust behavior. Artificial Intelligence, **172**(4):392–412.

NRF. 2014. Network robot forum. www.scat.or.jp/nrf/English/.

RASHIDI, P. 2009. The resident in the loop: Adapting the smart home to the user. IEEE Transactions on Systems, Man, and Cybernetics journal, Part A., **39**(5):949959.

ROCCO, M. DI, F. PECORA, P. KUMAR, and A. SAFFIOTTI. 2013. Configuration planning with multiple dynamic goals. *In* Proc. of AAAI Spring Symposium on Designing Intelligent Robots.

ROGGEN, DANIEL, KILIAN FORSTER, ALBERTO CALATRONI, and GERHARD TROSTER. 2013. The adARC pattern analysis architecture for adaptive human activity recognition systems. Journal of Ambient Intelligence and Humanized Computing, **4**(2):169–186.

S. CORADESCHI ET AL. 2013. GiraffPlus: Combining social interaction and long term monitoring for promoting independent living. *In* Proc of the 6th Int Conf on Human System Interaction, IEEE, pp. 578–585.

SAFFIOTTI, A., M. BROXVALL, M. GRITTI, K. LEBLANC, R. LUNDH, and J. RASHID. 2008. The peis-ecology project: Vision and results. *In* In Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS) - to appear, 2008. In: Proc. of the IROS-08 Workshop on Network Robot Systems.

SANFELIU, A., N. HAGITA, and A. SAFFIOTTI. 2008. Special issue on network robot systems. Robotics and Autonomous Systems, **56**(10):793–797.

TANG, FANG, and LYNNE E. PARKER. 2005. ASyMTRe: Automated synthesis of multi-robot task solutions through software reconfiguration. *In* Proc. of the 2005 IEEE International Conference on Robotics and Automation, ICRA 2005, pp. 1501–1508.

TAPIA, EM., SS. INTILLE, and K. LARSON. 2004. Activity recognition in the home using simple and ubiquitous sensors. *In* Pervasive Computing, pp. 158–175.

VIG, LOVEKESH, and JULIE A ADAMS. 2006. Multi-robot coalition formation. Robotics, IEEE Transactions on, **22**(4):637–649.

WATKINS, C.J.C.H. 1989. Learning from delayed rewards. Ph. D. thesis, University of Cambridge, England.

YILDIZ, I.B., H. JAEGER, and S.J. KIEBEL. 2012. Re-visiting the echo state property. Neural networks, **35**:1–9.

ZHANG, SHUAI, S.I. MCCLEAN, and B.W. SCOTNEY. 2012. Probabilistic learning from incomplete data for recognition of activities of daily living in smart homes. Information Technology in Biomedicine, IEEE Transactions on, **16**(3):454–462. ISSN 1089-7771. .

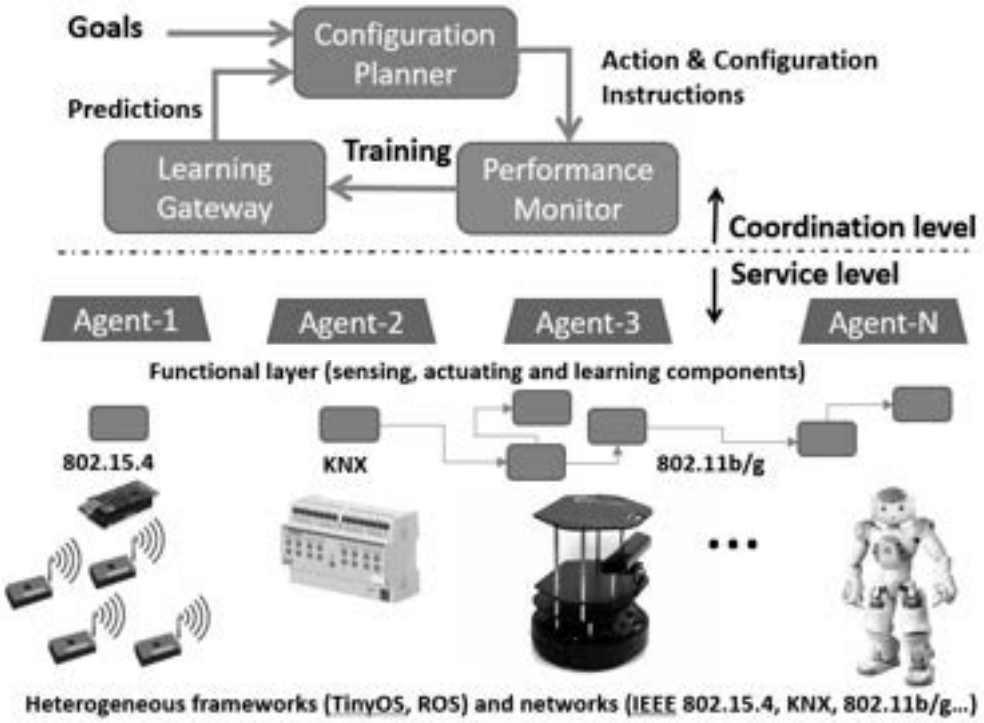


FIGURE 1. Sketch of the system architecture illustrated in this section

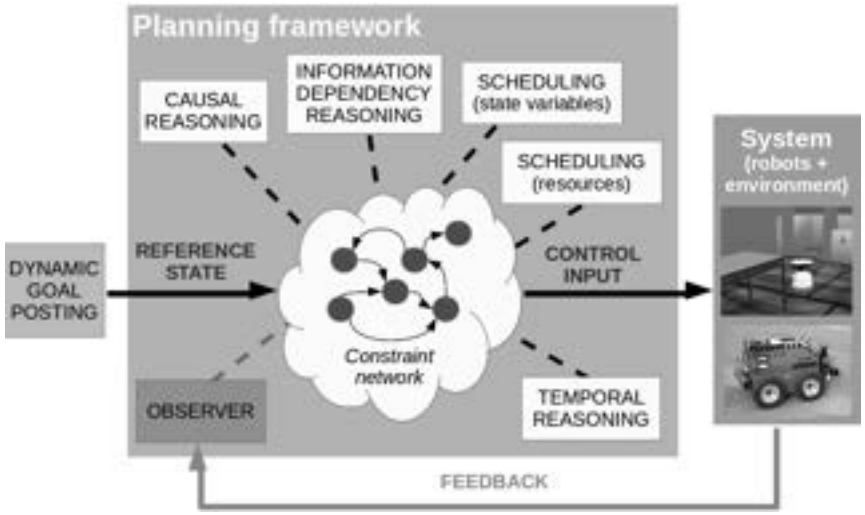


FIGURE 2. Schematic view of the configuration planner. The observed state and the current goals are continuously injected in the shared constraint network. Several solvers (yellow boxes) complete this network with a plan that connects the current state to the goals.

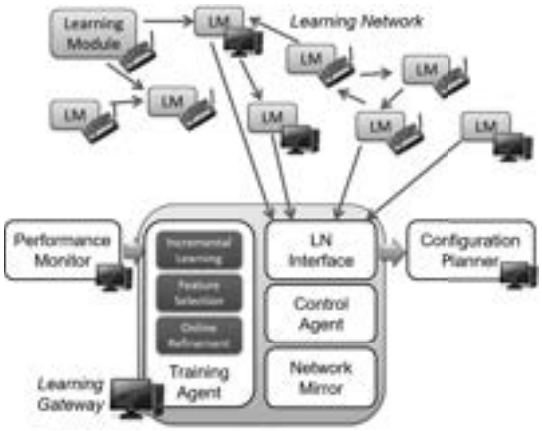


FIGURE 3. Architecture of the distributed learning system highlighting interactions with the *Performance Monitor* and *Configuration Planner* instances. The picture shows the Learning Modules distributed on devices with different computational capabilities (motes and PC icons). Relevant software agents hosted by the Learning Gateway are represented as white boxes: the Training Agent interfaces with the performance monitor and performs computationally intensive learning and self-configuration routines with the support of a mirrored copy of the distributed learning network (Network Mirror). The Control Agent component manages the whole learning system, while the LN interface collects and dispatches the predictions computed by the distributed Learning Network.

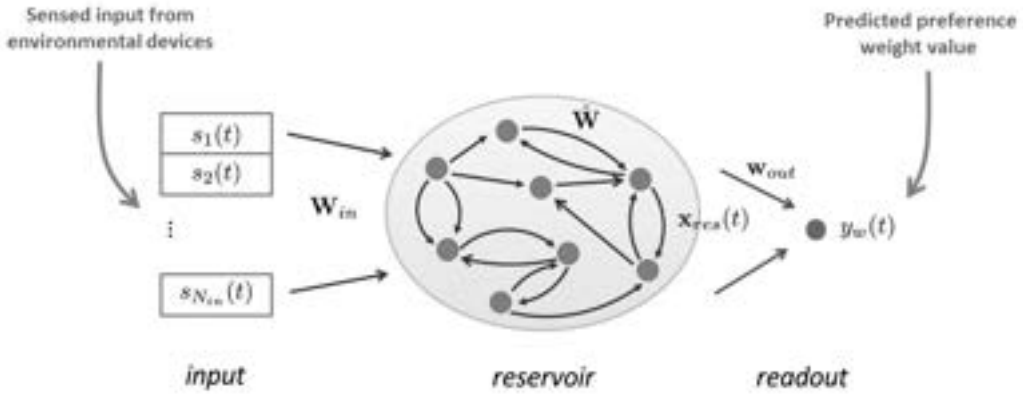
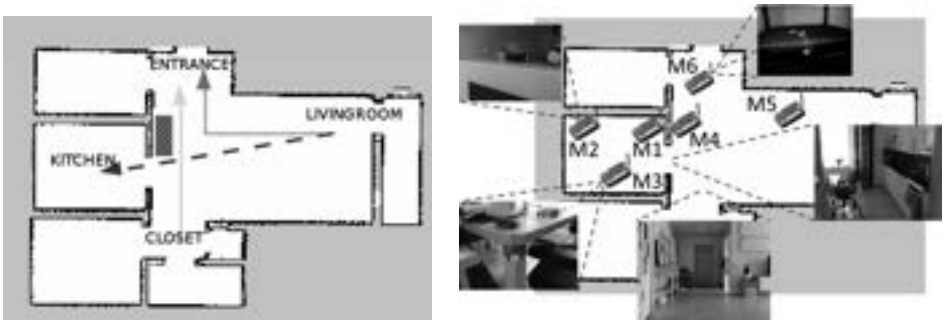


FIGURE 4. Graphical illustration of the architecture of a LI-ESN learning module. Streams of sensed input from environmental devices are processed by the reservoir component, which provides a dynamical memory to the system. Predictions about the preference weight values are computed by the readout part as output of the LI-ESN. Only connections in red require to undergo a training process (see text).



(a) (b)

FIGURE 5. Map of the Ängen senior residence flat (resulting from SLAM reconstruction) with highlighted mobile robot navigation tasks (a) and location of the WSN motes (b).

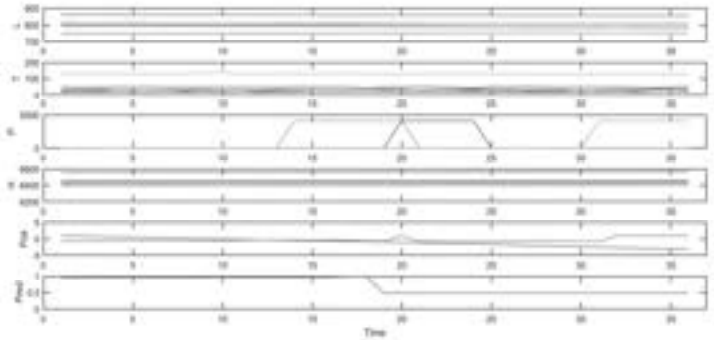


FIGURE 6. Example of sensor data grouped by sensor types (top 4 plots), trajectory information (fifth subplot) and ground truth performance values (bottommost plot) for the Entrance Mirror experiment.

FIGURE 7. Snapshot of the robot performing a navigation task towards the flat entrance: on the bottom-right, note the mirror reflecting the Kinect laser.

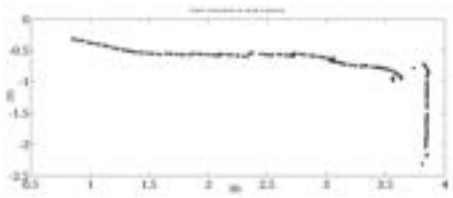


FIGURE 8. Localization estimating the path from the living-room to the entrance: during the turn the localization estimate is coarser since a mirror influences the data gathered by the Kinect sensor. Units are expressed in meters.

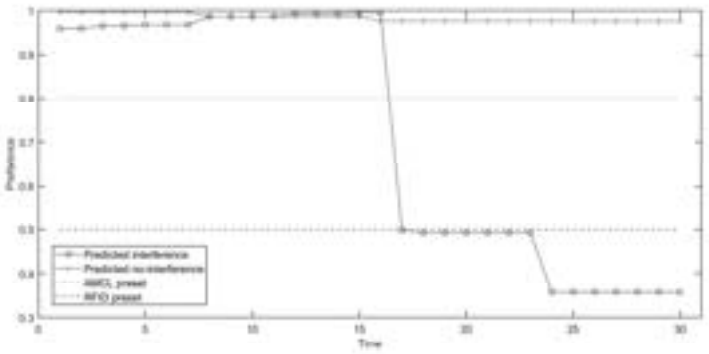
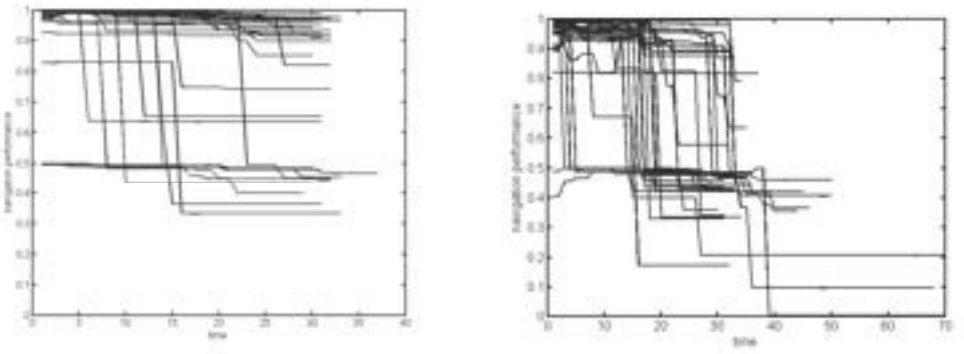


FIGURE 9. Fixed preference weights for the RFID and AMCL localization compared with two examples of adaptive AMCL preference weights, under disturbance conditions and without localization interference.



(a) (b)

FIGURE 10. Target functions for the straight (top) and curved (bottom) trajectories of the Entrance Mirror task: the latter are influenced by the presence of the mirror, while the former are not. Influence of the mirror is clear as the majority of the curved trajectories have a target value ending under 0.5 (which is the threshold value for determining if navigation is satisfactory).

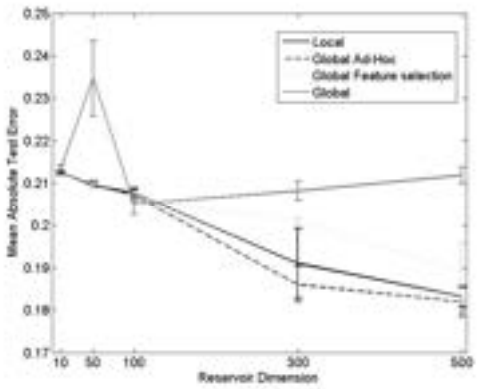
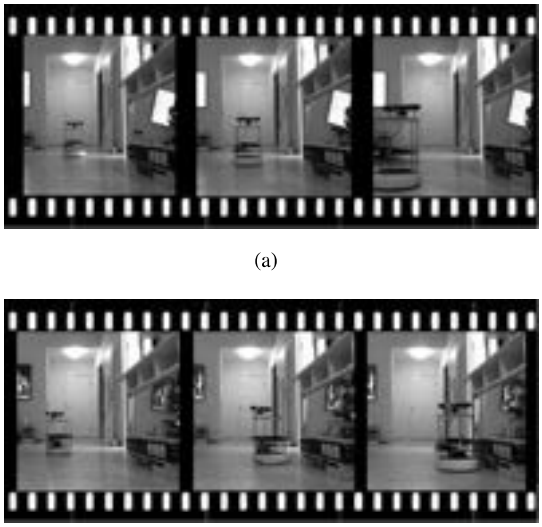


FIGURE 11. Test set performance (as Mean Absolute Error) on the Entrance Mirror task as a function of the number of reservoir neurons.



(a)

(b)

FIGURE 12. Frames from the accompanying videos of the online tests on the Entrance Mirror experiment: (a) AMCL navigation is unaffected by the mirror when the robot travels near the mirror arriving from the bedroom (straight trajectory); (b) the robot arriving from the living room (curved trajectory) gets confused near the mirror when navigating with the AMCL;

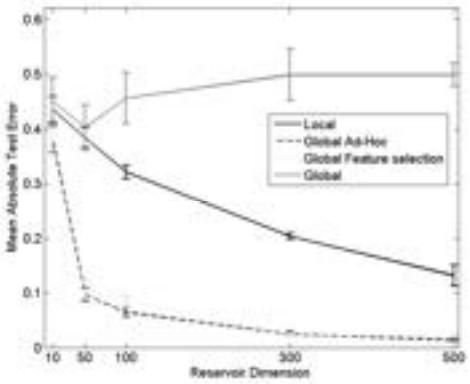


FIGURE 13. Test set performance (as Mean Absolute Error) on the Kitchen task as a function of the number of reservoir neurons.

FIGURE 14. Frames from the accompanying videos of the online tests on the Kitchen Cleaning experiment: the robot stops on its way to the kitchen after the user entered the room and waits until the user has left before proceeding with the cleaning.

TABLE 1. Input configuration for the Entrance Mirror experiment: input sources include robot x, y coordinates and orientation θ; for each WSN mote available transducers include L (light), T (temperature), H (humidity), P (PIR).

Experimental Setting	**ESN input configuration**
local	robot: (x, y); M_6: P
global ad-hoc	robot: (x, y); M_3, M_6: P
global feature selection	robot: (x, y); M_1, M_2, M_3, M_6: P
global	robot: (x, y, θ); $M_1 - M_6$: L, T, H, P

TABLE 2. Input configuration for the Kitchen Cleaning experiment: input sources include robot x, y coordinates and orientation θ; for each WSN mote available transducers include L (light), T (temperature), H (humidity), P (PIR).

Experimental Setting	**ESN input configuration**
local	robot: (x, y); M_3: P
global ad-hoc	robot: (x, y); M_2, M_3: P
global feature selection	robot: x; M_1, M_2, M_3, M_4, M_5: P
global	robot: (x, y, θ); $M_1 - M_6$: L, T, H, P

TABLE 3. Performance of the learning model configurations selected at model validation phase for the Kitchen task. Performance is measured as MAE averaged over the 10 reservoir guesses; deviation is reported within brackets.

Experimental Setting	**Training MAE**	**Validation MAE**	**Test MAE**
local	$0.217101(\pm 0.006791)$	$0.374327(\pm 0.026519)$	$0.203766(\pm 0.005922)$
global ad-hoc	$0.004223(\pm 0.000426)$	$0.095127(\pm 0.033478)$	$0.013718(\pm 0.001649)$
global feature selection	$0.000723(\pm 0.000095)$	$0.101096(\pm 0.034971)$	$0.020300(\pm 0.002820)$
global	$0.040273(\pm 0.001900)$	$0.139266(\pm 0.048193)$	$0.498551(\pm 0.021220)$