UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior
Departamento de Informática

TESIS DOCTORAL

Building Planning Action Models Using Activity Recognition

Doctorado en Ciencia y Tecnología Informática

Autor

Javier Ortiz Laguna

Directores

Angel García Olaya y Daniel Borrajo Millán

Leganés, Junio 2014

Universidad Carlos III de Madrid

TESIS DOCTORAL

Building Planning Action Models Using Activity Recognition

Autor

Javier Ortiz Laguna

Directores

Angel García Olaya y Daniel Borrajo Millán

Departamento de Informática
Escuela Politécnica Superior

Leganés, Junio 2014

TESIS DOCTORAL

Building Planning Action Models Using Activity Recognition

Autor:	Javier Ortiz Laguna
Directores:	Dr. Angel García Olaya Dr. Daniel Borrajo Millán

Firma del Tribunal Calificador:

	Firma
Presidente: D.	....
Vocal: D.	....
Secretario: D.	....

Calificación: ....

Leganés, ... de .... de ....

A mi familia y amigos

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank Daniel Borrajo and Angel García Olaya for supervising my PhD studies. Thanks also to Hector Muñoz and Fernando De La Torre –visiting your research groups was a great experience.

Thanks to all the PLG members because I enjoyed working with you. Of course, thanks to Julio, Rodolfo, Borja, Gerardo, Juanjo and to all the people who practiced Aikido with me during all these years. You do not know it but you helped me a lot.

Special thanks to all the people that contributed to this work in one way or another.

I dedicate this thesis to the students who have been put off university by the tuition fees rise.

Resumen

El campo del reconocimiento de actividades realizadas por las personas recibe en la actualidad una especial atención debido a sus numerosas áreas de aplicación y al desarrollo de las tecnologías que lo hacen posible. En los últimos años, se ha investigado mucho en este área de la Inteligencia Artificial para tareas como el seguimiento de personas que presenten algún tipo de dependencia. Por ejemplo, personas mayores con demencia senil o cualquier otra discapacidad cognitiva. El reconocimiento de las actividades que estas personas llevan a cabo podría permitir la monitorización inteligente de dichas actividades y, en caso de ser necesario, asistir a estas personas para permitirles completar con éxito las tareas que deben realizar.

En la actualidad, existen sistemas capaces de reconocer lo que los usuarios de dichos sistemas realizan dentro de un entorno definido. La mayoría de estos sistemas tiene dos problemas. El primer problema es que reconocen sólo estados parciales de cada actividad. Es decir, en lugar de reconocer una actividad completa que comienza en el punto a y termina en el punto b, estos sistemas dividen la línea temporal en ventanas de tiempo de un tamaño determinado que clasifican como pertenecientes a una actividad u otra. Estas ventanas temporales tienen el problema de que muchas veces solapan dos actividades, lo que hace muy difícil la correcta clasificación de las actividades a las que pertenecen dichas ventanas. Además, esto también dificulta la obtención de los estados que anteceden y suceden a las actividades. Dichos estados son necesarios a la hora de construir modelos de comportamiento. El segundo problema es que reconocen actividades de alto nivel completas como *cocinar* o *preparar té* pero no las actividades de bajo nivel o acciones, como *coger el tenedor* o *encender la tetera*, que componen dichas actividades de alto nivel. Así, salvo unas pocas excepciones, la mayoría de los sistemas presentes en la literatura no pueden ser utilizados para asistir a los usuarios durante la realización de las actividades ya que no pueden saber cómo de completa está la actividad.

Por todo ello, en esta tesis se plantean tres grandes objetivos. El primero es la **implementación de un nuevo algoritmo de reconocimiento de actividades** que evite los problemas que provocan las ventanas temporales de longitud fija y pueda ser utilizado también para extraer los estados por los que el usuario lleva al sistema a través de sus acciones.

El segundo objetivo es la **generación automática de un dominio de planificación automática** que represente el comportamiento del usuario a partir de las actividades reconocidas. Para ello se utilizará el algoritmo desarrollado en el paso anterior para reconocer las acciones que componen las actividades y los estados que anteceden y suceden a cada acción. Una vez que se tiene un sistema capaz de generar los estados y acciones realizadas por el usuario, se genera un dominio de planificación utilizando dicha información. Entonces, el dominio de planificación podrá ser utilizado por un planificador automático ya existente para generar secuencias de acciones a partir del estado actual, que podrán ser

utilizadas para asistir al usuario en determinadas situaciones.

El tercer objetivo es el de **estudiar la utilización de los dominios de planificación generados** para crear planes y guiar con éstos a los usuarios del sistema. Además, se quiere comprobar si los dominios de planificación generados por el sistema pueden ser utilizados para reconocer actividades por sí solo o junto con un sistema de sensores para conseguir así mejores resultados. También se quiere probar su capacidad para predecir futuras acciones.

Abstract

Activity recognition is receiving a special attention because it can be used in many areas. This field of artificial intelligence has been widely investigated lately for tasks such as following the behavior of people with some kind of cognitive impairment. For instance, elderly people with dementia. The recognition of the activities that these people carry on permits to offer assistance in case they need it while they are performing the activities.

Currently, there are many systems capable of recognizing the activities that a user performs in a specific environment. Most of these systems have two problems. First, they recognize states of the activities instead of the entire activity. For instance, instead of recognizing an activity that starts at the moment *a* and ends at *b*, these systems split the time line in fixed-length temporal windows that are classified as belonging to an activity or another. These windows sometimes overlap two activities which makes classifying the activities more difficult. Also this prevents the system from detecting the states of the system before and after each activity. These states are needed to build behavioral models. Second, most of these systems recognize complete high-level activities such as *cooking* or *making tea* but they can not recognize the low-level activities that compose the high-level activities. For example, *pick-up the fork* or *switch on the oven.* For this reason, most of the systems in the literature can not be used to assist people during the activities since they recognize the activity itself and they can not provide the low-level activities that the user has to execute to complete the high-level activity.

For these reasons, in this thesis we have three objectives. The first objective is the **development of a new activity recognition algorithm** capable of overcoming the problems that the fixed-length temporal windows cause and, also, capable of extracting the states that the system can traverse.

The second objective is to **automatically generate an automated planning domain** able to represent the user behavior using the activity recognition system. In order to do that, we will use the activity recognition system developed in the previous step to recognize the activities and the states of the environment before and after the activities. Once the system is capable of performing this task, the planning domain is generated using that information. Then, the automated planning domain will be used by an automated planner to generate sequences of actions to reach the goal of the user. That way, those sequences will be used to assist users by telling them the next action or actions to accomplish their goals.

The third objective is **to use automatically generated planning domains for guiding users** to accomplish the task they pursue. In addition, we want to check whether the generated plans can be used to recognize the activities alone or to help a sensors system to improve its results. Also, the generated plans will be used to predict the next activities that the user may perform. This way, we will test the planning domains and the plans generated by the planner to check if they are capable of offering information to recognize the

activity that the user performs or, at least, offering information for the activity recognition system to improve its results.

Part I

Introduction and State of the Art

Chapter 1

Introduction

This thesis work belongs to the areas of Human Activity Recognition and Automated Planning in Artificial Intelligence. More specifically, it approaches the task of generating planning action models from sensor readings through activity recognition methods.

1.1 Overview

From the beginning, research on Artificial Intelligence (AI) had the goal of automatically solving problems. Among all the subdisciplines of AI we can find Automated Planning (AP) and Activity Recognition (AR). AP has been successfully used in many domains like robotics or transportation logistics. It aims at generating plans, or sequences of actions, for the purpose of achieving a particular goal. Thus, given a goal state, an initial state, and a planning domain, the actions that can be executed, an automated planner generates a plan that, when executed, will reach a goal state from the initial state. However, building an action model is a difficult and time-consuming task even for domain experts.

On the other hand, the field of AR is receiving great attention at the moment due to the wide range of applications in many areas of science and engineering. AR aims to infer the actions of one or more agents from a set of observations captured by sensors. The applicable technology improves fast. Among other advances, it allows now wireless sensors.

Advances in miniaturization technologies and communications have led to the creation of a new computing paradigm called Ubiquitous Computing (UC). The UC idea envisioned by Mark Weiser [Weiser, 1995] is also described as Pervasive Computing or Ambient Intelligence (AmI), where each term emphasizes slightly different aspects. Thus, UC pretends to incorporate small, inexpensive, robust and networked processing devices in everyday life. That way, a new human-computer interaction model is created where the processing power is distributed in the objects of the environment. On the other hand, AmI refers to electronic environments that are sensitive and responsive to the presence of people. In an AmI environment, the devices work in concert to support people in carrying out their everyday life activities using information that is hidden in the network connecting these devices [Ducatel *et al.*, 2001].

A system that uses such a computing model is based on a network of distributed processing devices connected to a communication network composed of sensors and communication protocols capable of capturing, processing and distributing context information about the users of the smart environment.

AR is closely linked to UC since it uses the information gathered by the sensor network as the starting point to recognize what the user is doing at a specific moment. It can be used in a wide range of applications such as context-aware computing or real-time information systems in order to assist people with some kind of impairment. Thus, it uses UC but also it can be part of the UC system making the interaction between the user and the system possible.

On the basis of these premises and using the principles of UC, this thesis proposes a new use of the technologies described above in order to incorporate assistance into a smart domestic environment. This thesis aims to use AR to automatically build AP domains and employ such domains to assist people with some kind of cognitive impairment in their homes. All the necessary algorithms will be implemented in an integrated system. Such system should be capable of providing personalized assistance for users to perform domestic everyday tasks. Specifically, in this thesis a system capable of detecting low-level and high-level activities that users perform is developed. Once the activities have been detected and recognized, they will be used to build an action model representing the user behavior. This way, the system will overcome the problems that building an action model for AP entails. Then, the system will be able to use the learned action model to assist the user, suggesting the best activity or activities to accomplish the user's goals. Also, it will be used to predict the possible next activities that the user will execute.

There are many problems to solve in order to develop such a system. The main one is the recognition of the activities that users perform in the environment. This is the axis this thesis turns around. The system will monitor users and check if they are able to accomplish the activities correctly. Otherwise, the system could show warnings in order for the users to successfully reach their goals.

The system begins with the sensor network deployed in an environment and also worn by the users. The sensor network will gather information of the actions of the user and the changes in the environment to infer the activity that the user performs. There are several types of sensors used to accomplish such a task and, besides, the readings of the sensors contain noise. Data Fusion is the process of integrating multiple data, recorded from a multiple sensors system, together with knowledge representing the same real-world object into a consistent and useful representation. So, in the first phase of this thesis, some data fusion techniques are used to process and filter the sensor data and solve the problems caused by the noise of the sensors and also to fuse the data produced by the sensor network. That way a unique source of data is generated.

Once the sensor data is processed, methods for time series analysis are used to extract the most relevant features for AR. Then, these features will be employed by machine learning techniques in order to infer the activities performed by the users. This way, a model of the activities executed by the users is generated to infer the activities later on. In addition to standard activity recognition, some researchers have differentiated a sub-area of the AR that they call *Activity Spotting*. This sub-area of the AR tries to infer very short activities or activities with no duration. These activities can be referred to as low-level activities or actions. This sub-area will be considered as well since most of the activities that are going to be recognized in this thesis have a very short duration or no duration at all. Specifically, most of the activities that are going to be recognized by the system developed in this thesis are actions that compose the activity of *cooking a recipe*.

The model generated in the AR step permits the system to know the actions that the user performs at a specific moment. This allows the system to perceive the current state of the user and the environment.

The next step of the thesis will be the generation of a user model, using AP, from the sequences of actions that the user performs and the state of the environment before and after every action.

The generated AR user model will provide the state of the system at a specific moment and the AP domain will be used to generate plans from every single state for the user to accomplish his/her goal. This way, with a planning domain capable of reproducing the user behavior, the developed system will be able to assist the user of the system by providing the next action that the user should execute in order to reach his/her goal.

As final objective, this thesis will evaluate the created system to check if the generated user action models coded as planning domains are capable of helping the system for recognizing activities. This way, the activity recognition system and the automated planning system will compose a new activity recognition system.

Figure 1.1 summarizes the tasks that have to be implemented to develop the proposed system as well as the technologies and the research fields involved.

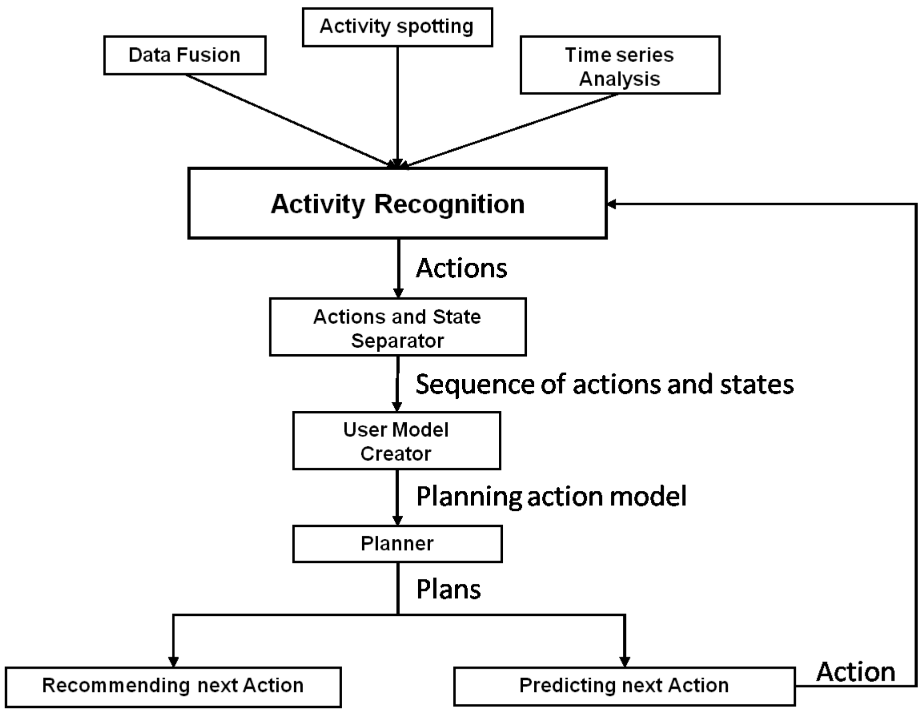

Figure 1.1: Scheme of the research fields involved in the thesis and the modules that compose the whole system.

1.2 Reader's Guide to the Thesis

This document is organized in four parts: **Part I** introduces the thesis and describes the state of the art. Chapter 2 reviews the literature about activity recognition and classical planning. **Part II** details the objectives that this thesis pursues along with the evaluation method employed to analyze the results obtained. **Part III** explains the work done along the thesis research. Particularly, it describes work in activity recognition using a relational model (Chapter 5); Chapter 6 analyzes a new way of segmentation and present models based on that idea; Chapter 7 presents a complete activity recognition algorithm employing the segmentation method proposed in Chapter 6; and Chapter 8 details a method to learn planning action models using the information provided by sensors through an activity recognition system. This method integrates activity recognition and planning to build a system capable of providing assistance for the user to reach a goal. Finally, **Part IV** summarizes the contributions of this thesis and discusses some conclusions and future work.

Chapter 2

State of the Art

This chapter introduces the theoretical foundations that are the basis of this work as well as a description of the main research areas involved in the development of the proposed system. We begin with a section about AR and all the technologies used, including sensors and temporal series analysis. Next, different modelling approaches for the classification of the activities are presented. Then, AP is introduced along with some approaches for automatically generating planning action models.

2.1 Activity Recognition

Activity Recognition aims to automatically infer the actions and goals of one or more persons from a series of observations gathered by a sensor network. The sensor network may be composed of one or more sensors of different types. So, the sensor network will capture information of diverse nature depending on the types of sensors used. Thus, AR attempts to infer high-level information about the activities that users perform using the low-level data provided by the sensors. This research field is the centre of this thesis.

The first works in the field of AR started in the 1990s when the advances in electronics technology enhanced the computer, communication and sensor equipment. This permitted to develop equipment small and light enough to be integrated in mobile systems capable of being worn by a person as any other piece of clothing during a long enough period of time [Starner *et al.*, 1999].

Although the first relatively bulky prototypes were far from being integrated in the environment as Mark Weiser predicted [Weiser, 1995], they kept the promise of making the computer to perceive the user's life in first person, permitting the creation of really personal applications. The first works focused on text and were based on the use that users made with a keyboard. Gradually, the systems explored new methods for gathering information and new interfaces for users. For instance, small portable cameras [Schiele *et al.*, 1999; Starner *et al.*, 1997] or microphones [Clarkson and Pentland, 1998]. Also, context information as the location of users, the topic of conversation or the identity of the persons with whom users spoke was incorporated. This information gave the user clues about his/her current situation in real-time or just store that information to be used later on [Rhodes, 1997]

Measuring the physical activity of a person through the use of technology has been the objective of part of the medical community for a long time; accelerometers have been used with this purpose for decades [Montoye *et al.*, 1983; Wong *et al.*, 1981]. These not very mature systems attempted to estimate measures such as energy expenditure or the oxygen consumption of the user while he/she was executing physical activities. At the end of the past decade, some mobile systems incorporated inertial sensors and were able to separate and recognize some specific physical activities stimulated by the technological advances and the development of relatively inexpensive communication, computation, and sensing devices. Such advances were produced in the equipment available in the marked but also in the machine learning methods [Golding and Lesh, 1999; Randell and Muller, 2000; Van Laerhoven and Cakmakci, 2000].

The current research in AR covers a wide range of applications. There are research groups focused on recognizing Activities of Daily Living (ADLs) in the context of health care and elderly care [Lester *et al.*, 2006]. Others focus on performing automatic AR using non-supervised or semi-supervised machine learning algorithms [Wyatt *et al.*, 2005; Huynh and Schiele, 2006; Minnen *et al.*, 2007] to avoid the problem of labeling data. Also, there are groups dedicated to combine different types of sensors to improve the results [Stiefmeier *et al.*, 2006; Wang *et al.*, 2007].

Other terms related to AR and even sometimes considered the same task by the scientific community are *Plan Recognition* and *Goal Recognition*. In the field of automated planning, plans, goals and activities are words used to define different terms. Thus, the goals are the states that the user wants to achieve, activities are the actions that users can execute to transform the environment and the objects in it to modify the current state and plans are sequences of actions that users execute to reach a goal.

From a practical point of view these terms could be divided into two groups depending on the problem that they resolve and the methods used. On the one hand, AR starts from the information gathered by a sensor network to infer the activities performed by users and, on the other hand, plan and goal recognition start from a sequence of actions and they attempt to infer the goals that an agent pursues and the next actions that this agent is going to execute to reach the goal. For AR, the starting point is the sensorial information and this data normally contains noise. So, in order to accomplish the recognition, techniques to deal with noise have to be used to permit the use of the information provided by the sensors. However, goal and plan recognition do not need to deal with the sensor noise so do not use those techniques. They use higher level information, the actions executed. Anyhow, some methods are shared by the three areas and, in the last years, the boundaries among them are becoming very diffuse. The three terms are defined here to clearly differentiate that they are different tasks and, therefore, they will be treated as different terms in this thesis.

2.1.1 Sensors

The types of sensors used for AR are very diverse. At the beginning, most works used vision sensor networks and audio sensors [Pentland, 1996; Gavrila, 1999] although the use of vision sensor networks has continued up to the present [Junejo *et al.*, 2011; Gómez-Romero *et al.*, 2012]. Later on, other types of sensors were gradually integrated. Thus, location sensors started to be used. In [Liao *et al.*, 2005], authors use GPS to detect activities performed in the street while [Yin *et al.*, 2004] use a WIFI network to infer the high-level goals of users from the low-level signals of a device connected to the WIFI network of an office.

Physical activities require repetitive movements of the human body or a specific bodily position. Therefore, in order to recognize this type of activities other types of sensors are employed. So, works like [Bao and Intille, 2004; Kahn *et al.*, 1999; Kasten and Langheinrich, 2001] use inertial and magnetic sensors as accelerometers, gyroscopes and magnetometers.

These sensors were used previously in robotics to measure the movements of robots.

Accelerometers have the capability of measuring linear acceleration and deceleration. A 3-axis accelerometer would ideally measure linear acceleration for the x, y, and z axes which are separated by exactly 90 degrees relative to the platform or object where the accelerometer has been installed. They also permit to know the relative position of an object with respect to the gravity field of the Earth. Gyroscopes measure the angular velocity. They are used currently, together with the accelerometers, to estimate the position and the location of an agent when it is not possible to use other location systems like GPS or when GPS does not provide accurate enough information. Magnetometers, however, used very often along with accelerometers and gyroscopes, are employed to measure the magnetic field of the Earth or magnetic fields near of the device. They are capable of determining the relative position of an object with respect to the magnetic north pole.

Other types of activities can be easier to recognize through the objects involved in the activity. For instance, the objects that the user touches or uses when executing some activities as cooking or shaving may provide better information than the way the person moves the arm. RFID technology provides remote data storage and retrieval systems that use devices called RFID tags or transponders. The fundamental objective of the RFID technology is to transmit the identity of an object (similar to a unique serial number) through radio waves [Finkenzeller, 2010]. The RFID tags are small devices, similar to a sticker in some cases, which may be attached to an object, animal or person. RFID tags contain at least two parts: an integrated circuit for storing and processing information, modulating and demodulating a radio-frequency signal, collecting power from the incident reader signal, and other specialized functions; and an antenna for receiving and transmitting the signal. Thus, this technology may permit the detection of the objects that a person holds or even it could be used to locate people [Ni *et al.*, 2004]. [Fishkin *et al.*, 2004] use RFID devices to detect motion and [Philipose *et al.*, 2004; Fishkin *et al.*, 2005; Patterson *et al.*, 2005] present AR systems that employ RFID sensors to identify the object that the users handle. Then, the system uses that information to infer the activities that were performed.

RFID sensors are not the only choice to detect the objects that a person uses. Other types of sensor can be used as it is shown in [Tapia *et al.*, 2004] where the authors employed wireless accelerometers to know which objects were utilized by the user and how they were employed. The accelerometers were attached to many objects and pieces of furniture to monitor the interaction of the user with the environment. Other works like [Pham and Olivier, 2009; Hoey *et al.*, 2011] attached wireless accelerometers just on few objects, those objects that the user could employ in order to perform specific activities. The accelerometers along with other types of sensors were employed to recognize the rest of the activities.

Other types of sensors that do not appear in the literature as often as RFID or accelerometers but employed in some works include wearable plastic optical fiber sensors for monitoring seated spinal posture [Dunne *et al.*, 2006], reed switch sensors to detect the state of doors, furniture or lights [Kasteren *et al.*, 2008], binary sensors that at any given time supply a value of one or zero like motion detectors, contact switches, break-beam sensors, and pressure mats [Wilson and Atkeson, 2005], garment-integrated foam-based pressure sensors used for monitoring the wearer's respiration rate [Brady *et al.*, 2005], wearable force sensors placed on the muscle surface for detecting and interpreting muscle activity and several types of physiological sensors like blood oximeter to monitor the user's blood oxygen level and pulse while sleeping [Oliver and Flores-Mangas, 2006], galvanic skin response, also known as electrodermal response, which measures changes in electrical resistance across two regions of the skin [Westeyn *et al.*, 2006], electrocardiogram sensors [Linz *et al.*, 2006], body temperature, combinations of the other sensors [Gerasimov, 2003] and mobile phones [Kwapisz *et al.*, 2011; Blázquez *et al.*, 2012].

Using the same type of sensors but in several places may improve the results. For instance, several accelerometers [Bao and Intille, 2004; Huynh *et al.*, 2008], integrating a set of different sensors in one device [Choudhury *et al.*, 2008] or using different types of sensors in the same network [Junker *et al.*, 2005; Kern *et al.*, 2004]. Complementary sensors combined in groups of two or more may also help to better recognize activities. For example, combining sensors to measure movement and localization [Subramanya *et al.*, 2006], movement and audio [Lukowicz *et al.*, 2004; Kern *et al.*, 2004; Choudhury and Pentland, 2003], movement and proximity [Stiefmeier *et al.*, 2006], movement and RFID [Wang *et al.*, 2007; Stikic *et al.*, 2008a] or many different types [Tapia *et al.*, 2004].

The last examples show the combination of wearable sensors and sensors installed in the environment, in this case RFID tags attached to some objects. The data employed in their systems is part of a bigger data set introduced in [Logan *et al.*, 2007] where authors employed portable sensors combined with a big group of sensors deployed in the environment to detect the use of the objects: motion sensors, magnetic sensors to detect changes in the state of doors, windows, pieces of furniture, etc; and to detect the environmental conditions: light, temperature, humidity, etc. [Logan *et al.*, 2007] gathered the data in a laboratory where a big set of different types of sensors were deployed to capture as much information as possible from the environment and from the users, in a place that simulates an average flat. In such flat-laboratory, a couple was living for several weeks [Intille *et al.*, 2006] to provide a data set as real as possible. In [Haya *et al.*, 2004] authors presented a context-based architecture to achieve the required synergy among the ubiquitous computing devices and it was also implemented in a real environment; a living room and an office space. They used sensors to detect presence, temperature, and luminosity, among other features.

2.1.2 Segmentation

The starting point of an AR system is the sensor network. From a set of sensor readings, the system has to extract higher-level information, features or attributes, in order to classify and match the sensors' readings and the performed activities. Thus, sensors produce time series that have to be processed in order to extract information.

A temporal serie may be defined as a time-ordered sequence of observations on a variable taken over time. So, time series analysis comprises methods for analyzing changes of a variable over time. In this thesis, time series analysis is important since the observations gathered from sensors create time series and these time series are the starting point of the system that is going to be developed.

AR systems extract relevant information from sensor time series in order to create action models to classify the activities. This information depends on the type of sensors used. So, there are systems that employ RFID readers and tags like [Philipose *et al.*, 2004] which generate as observations the identification of the RFID tags detected by the RFID reader. Reed switch sensors [Kasteren *et al.*, 2008] produce a two-states signal corresponding to the two states in which reed switch sensors can be. These two types of sensors generate readings with discrete values and events at specific moments of time.

Other types of sensors produce observations with continuous values. For that reason, the extraction of relevant information is more difficult. For instance, the systems that employ accelerometers or gyroscopes [Bao and Intille, 2004; Junker *et al.*, 2005]. In these cases, the

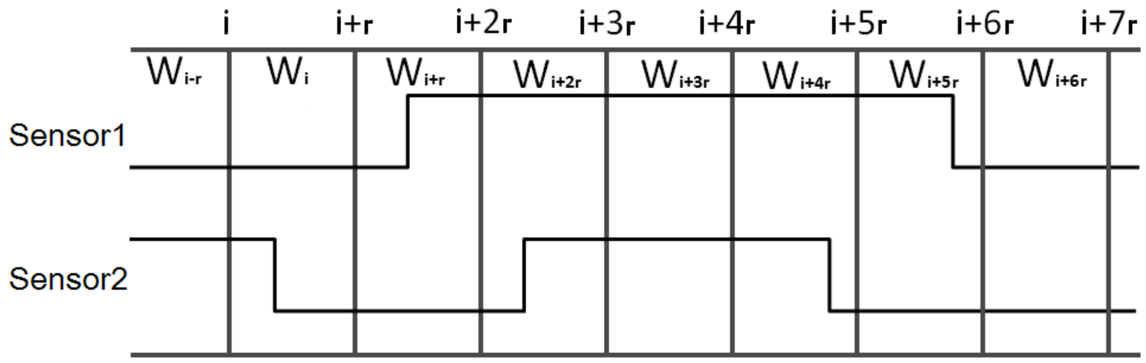

Figure 2.1: Temporal segmentation on time series of two sensors by the sliding window method.

systems usually extract statistical values like the mean or the variance or even the energy of the signal of an interval of values or the frequency after changing the readings from the time domain to the frequency domain. This is done using the Fast Fourier Transform (FFT) [Heideman *et al.*, 1985]. The time domain shows how a signal changes over time, whereas the frequency domain shows how much of the signal lies within each given frequency band over a range of frequencies.

For the extraction of these features, researchers have employed different techniques although the most widely used is the *Sliding Window* (SW).

A formal definition of an AR process can be defined as follows. Given a network of N sensors and a set A of n activities $A = a_1, \ldots, a_n$ that the user might perform at each time step, N sequences will be generated by an AR system where each sequence can be represented as a vector $X^s =< .., x_i^s, .. >$ of readings of sensor s. x_i^s is the sensor reading at time i of sensor s. So, the first task consists of defining a function f_1 that takes the N sequences and returns Z vectors of features $\vec{F_i}$. Each vector is labeled with the activity a that the user performed during the period of time from i to $i + l$ when the features in $\vec{F_i}$ were extracted. The second task is to learn a function f_2 that takes as inputs those vectors $\vec{F_i}$ produced by f_1 and builds a classifier to infer the activities performed.

2.1.2.1 Sliding Window

The static sliding window approach uses fixed-length temporal windows that shift to create instances. Each window position produces a segment that is used to isolate data for later processing. It uses two parameters: the windows length l and the shift r. Figure 2.1 shows an example of sliding windows where $l = r$, i is the timestamp at which the first window starts and $i + l$ is the timestamp at which it finishes and the next temporal window starts.

So, using this method, the function f_1 may be defined as follows. Given a network of N sensors, N sequences of data are generated. Each sequence is segmented in Z temporal windows or time slices of l seconds in length defined as $W_i^s =< x_i^s, ..., x_{i+l-1}^s >$ of contiguous readings from the sensor s starting at time i. The window shift, r, defines the next temporal window as $W_{i+r}^s =< x_{i+r}^s, ..., x_{i+r+l-1}^s >$. The segments that start at time i are grouped in the matrix $W_i =< W_i^1, .., W_i^N >$. These temporal windows are represented in Figure 2.1. Then, the features are extracted from W_i to build the vector F_i which is labeled with the activity $a \in A = \{a_1, a_2, ..., a_n\}$ that the user performed during W_i. Thus, given a set of

n activities $A = \{a_1, a_2, ..., a_n\}$ (classes), every temporal window W_i generates a vector $\vec{F_i}$ that is labeled with an activity $a \in A$. Then, the function f_2 builds a classifier to find the mapping between $\vec{F_i}$ and the activity in A that was performed by the user.

So, in order to use the SW method two parameters have to be defined: length of the window l and the shift of the window r.

In [Kasteren *et al.*, 2008] time series data are divided into time slices of constant length, $l = 60$ seconds, and $r = l$ and each slide is labeled with the activity. After that, probabilistic models were used to capture the mapping between the time slices and the activities. The authors of [Patterson *et al.*, 2005] aim to infer the activity being performed each second, $r = 1$ second, by using a window of data of $l = 74$ seconds; in their case, l was set to the mean amount of uninterrupted time spent performing an activity. In this case, they used overlapped windows shifting them one second.

[Bao and Intille, 2004] employ accelerometers to capture data about users' movements. They use a SW with $l = 512$ and $r = 256$. Given that the sensors were sampled at 76.25Hz, the window size was 6.7 seconds. Mean, energy, frequency-domain entropy, and correlation features were extracted from the SW signals for activity recognition. Others like [Tapia *et al.*, 2004] use SW with l equal to the average duration for each activity computed from all the activities and r was half of the duration of the quickest activity. In [Kern *et al.*, 2007], the authors employ SW of 0.5 seconds. In [Stikic *et al.*, 2008a], the sensors used in [Bao and Intille, 2004; Tapia *et al.*, 2004; Kern *et al.*, 2007] are combined. Each feature is computed over a sliding window shifted in increments of 0.5 seconds. They evaluated the performance of the features both individually and in combination, and over different window lengths (0.5sec-128sec).

Other very different works that use the same method are [Bulling *et al.*, 2009; Deleawe *et al.*, 2010]. Bulling *et al.* [2009] use temporal windows to recognize eye gestures and Deleawe *et al.* [2010] use fixed windows of time of different sizes to predict CO_2 levels as an indicator of air quality in smart environments.

In [Huynh and Schiele, 2005], different features and window lengths are studied to recognize some activities. They show that the best performance is achieved when different window lengths and features are chosen separately for each activity.

2.1.2.2 Other Approaches

Although the static-length sliding-window is the most commonly used method, it is not the only one. In [Amft *et al.*, 2007] the authors split activities, that they call composite activities, into actions, that they call atomic activities. First, they recognize the actions through what they call detectors, autonomous sensors capable of recognizing some specific actions. Every detector recognizes one of several actions and reports those actions as events of the system. The events are used to recognize every activity. In [Junker *et al.*, 2008], the same authors also split activities in parts in a very similar way to [Amft *et al.*, 2007]. They use a two-stage method which consists of a pre-selection stage, which aims to localize and preselect sections in the continuous signals, and a second stage that classifies the candidate sections selected in the previous stage. The pre-selection stage looks for relevant motion events which select intervals of sensor readings that are classified in the second stage.

Another approach based on events was used in [Modayil *et al.*, 2008], where they used all readings reported by the sensor, an RFID reader, and they generate temporal windows using as boundaries the sensor readings. It is a kind of SW but using the readings to establish the length and shift of the windows.

A very different approach was presented in [Kerr *et al.*, 2011] in which the authors use finite state machines to recognize activities in virtual worlds. In this work, Kerr *et al.* used boolean propositions. They represent activities as a *propositional multivariate time series* task instead of a sensor reading task. Then, they try to find Allen's temporal relations [Allen, 1983] among propositions; finally, they build a finite state machine, using as states the relations among propositions found in each activity to classify.

2.1.3 Algorithms for Activity Modeling

Once the segmentation has been performed and the features extracted from the sensor readings, machine learning methods are employed to associate the features extracted and the activities. Many are the machine learning methods employed for AR in the last few years. Among them, generative models like Bayesian Networks and a wide variety of discriminative models like Decision Trees, k-Nearest Neighbor (kNN) or Support Vector Machines (SVM), as well as sets of classifiers. In this section, we will describe some methods in the literature.

2.1.3.1 Supervised Models

2.1.3.1.1 Probabilistic Models

Probabilistic models have been widely used in fields like speech recognition, signal coding, computer vision or activity recognition. At present, the use of these models is increasingly growing and they are, without any doubt, the most widely used techniques. This is due to the fact that probabilistic models have the ability to naturally manage the randomness of the activities performed by humans.

A probabilistic model (or stochastic) is represented by a distribution of probabilities for all the possible results. They employ statistical techniques for estimation, testing and prediction. These models are widely used in the field of AR since sensors produce significant amounts of noise and the activities performed by users are executed in a non deterministic way. Next, different probabilistic models present in the literature are discussed.

Dynamic Bayesian Networks

One of the most widely used methods for AR are Dynamic Bayesian Networks (DBNs). DBNs are derived from Bayesian Networks to incorporate the temporal aspect of a process in a more effective way [Singer and Warmuth, 1999]. DBNs are generally used to model Markov processes. A Markov process is a discrete stochastic process in which the past is irrelevant to predict the future given the current state. At each time step, only the variables in the current time step are employed to calculate the state of the variables in the next time step. Bayesian networks are directed acyclic graphs where the nodes represent variables, and the arcs stand for relationships among the variables. The inference in this model can be completed iteratively with only two sets of variables: one representing the beliefs at the previous time step, and the other representing beliefs at the current time step.

DBNs have been used to model human activities in various scenarios. For instance, the work done in [Patterson *et al.*, 2005] employs DBNs along with other probabilistic models. It attempts to recognize routine activities performed normally in a kitchen like *making tee* or *setting the table*. [Yin *et al.*, 2004] uses DBN models with multiple layers to recognize similar activities, while [Patterson *et al.*, 2004a] use DBNs to recognize (ADL), in the same way as [Philipose *et al.*, 2004].

Hidden Markov Model

Hidden Markov Models (HMMs) are statistical Markov models in which the system being modeled is assumed to be a Markov process with unobserved or hidden states. The objective is to determine the unknown or hidden parameters using the observable parameters. The observed parameters can be used to execute further analysis, for instance in applications like pattern recognition. A HMM can be considered as an instantiation of a DBN. HMMs have so far been the most used model for activity modeling.

These models have shown excellent performance in applications such as speech recognition [Rabiner, 1989] and they have also been widely used in AR as shown in [Patterson *et al.*, 2005; Kasteren *et al.*, 2008; Ward *et al.*, 2006; Oliver *et al.*, 2002] among many other works. The most commonly used approach is to train a HMM model using the Baum-Welch algorithm [Rabiner and Juang, 1993]. Another option for AR using HMMs is to use a HMM where each node represents one of the activities to recognize [Kasteren *et al.*, 2008]. In this case, the sequences of activities can be inferred using the Viterbi algorithm [Rabiner and Juang, 1993] or Particle Filters [Ristic *et al.*, 2004]. [Patterson *et al.*, 2005] shows an example of the two possibilities plus a third possibility that uses the inner states of the HMM to represent each activity along with the observation obtained in the state.

Conditional Random Fields

Conditional Random Fields (CRFs) is a statistical modelling method very often employed for labeling or parsing sequential data or extracting information from documents. In computer vision, CRFs are often used for object recognition and image segmentation. In some contexts they are also called Markov Random Fields.

[Kasteren *et al.*, 2008] presented a comparison between the performance obtained using HMMs and CRFs for ADLs recognizing. [Liao *et al.*, 2007] used CRF to infer external activities using GPS. Other related application but a bit far is the one shown in [Vail *et al.*, 2007] where CRFs are used to recognize the activities performed by robots.

Probabilistic Grammars

A probabilistic grammar is a grammar in which each rule has a probability assigned. So, the probability of appearance of a phrase is the product of the rules used to form such phrase [Suppes, 1970]. The simplest probabilistic grammars are the Context-Free Grammars (CFG). Thus, Probabilistic Context-Free Grammars (PCFG) are employed in areas as diverse as Natural Language Processing or the study of RNA molecules in Bioinformatics.

Standard algorithms for syntactic analysis can be used to infer the most likely plans that explain the observed sequences. However, the validation of these algorithms is difficult in practice due to the need of complete sequences for inferring. Incomplete sequences can not be used and, normally, the sequences of observations are rarely complete. For that reason, [Pynadath, 1999] solves the problem transforming the PCFG in a bayesian network for recognition. Another problem present in PCFG is that they are very restrictive since they do not keep the information about the current state of the agent. A way of solving the problem is the use of context-sensitive grammars but they rapidly lead to intractable complexity. Another alternative presented in [Pynadath and Wellman, 2000] utilizes what they called Probabilistic State-Dependent Grammars (PSDG). PSDG introduce variables in order to represent the states of the world and the state of the agents and, at the same time, they use structures to keep the inference process as simple as possible.

In the above mentioned cases, probabilistic grammars were used to recognize sequences of actions, plan recognition. However, these models have also been used in [Bobick and Ivanov, 1998] for AR, where authors used HMM to infer low-level events that were used to form an alphabet. Such alphabet was the input for a PCFG employed for activity and behavior recognition.

2.1.3.1.2 Non-Probabilistic Models

Although most works use probabilistic models, other works used different techniques to address the problem.

Grammars

[Ryoo and Aggarwal, 2006] describe a general methodology for the recognition of complex activities using context-free grammars to represent composite activities and their interactions.

Decision Trees

[Lombriser *et al.*, 2007] attempt to create a system for real-time activity recognition over their sensor network platform. In order to evaluate their system, they generated several models and one of them is based on decision trees. [Bao and Intille, 2004] developed an algorithm for recognizing physical activities through five accelerometers. As in the previous case, they used decision trees along with other models. Similarly, [Logan *et al.*, 2007] also used decision trees along with a bayesian network for AR in a laboratory where a flat was built and provided with a huge and heterogeneous sensor network.

Logic Models

Actions models based on logic have a long history. The model of Kautz [Kautz, 1991] based on events hierarchies was one of the first models used to infer action models. His model employed first order logic to represent the relations between actions. However, the model does not consider uncertainty.

In [Goldman *et al.*, 1999], Goldman *et al.* formalized AR as a Probabilistic Horn Abduction (PHA) problem [Poole, 1993]. PHA employs rules like those used in PROLOG to distinguish among several hypotheses. In the field of AR, the hypotheses are possible activities that explain the observations. They showed that it is capable of handling situations that cause problems to other researchers using his method.

All these examples belong to the area of Plan Recognition. In AR, there are not many references to the use of logic models. Among them, we can find [Landwehr *et al.*, 2008], where authors develop a labeling system based on the principles of the Inductive Logic

Programming (ILP). Other work is presented in [Dubba *et al.*, 2010], where a supervised framework is developed to learn clausal event models from large datasets using ILP. They used deictic spatial and temporal terms to provide positive and negative examples for an event for learning their models. Logic is used for representing the tracking data using a tree structured type hierarchy.

Hand-crafted Models

All models described above were learned automatically. Other researchers used hand-crafted models. Among the works that employed this kind of models, [Fogarty *et al.*, 2006] utilized microphones to recognize home activities related with water consumption. This work presents an inference system built from the activation patterns of the microphones. Such patterns were handcrafted using training data captured for that purpose. Also, in [Hong and Nugent, 2010] its authors show a model based on location sensors. This work proposes an algorithm that detects the beginning and end of each activity from the locations of the user captured by the sensor network. Once the boundaries of the activities were detected, the activity was identified. This work employed an ontology to represent activities and the interaction of the object with the environment in each activity.

Other Approaches

[Lombriser *et al.*, 2007] and [Bao and Intille, 2004], in addition to decision trees, also generated models using the kNN algorithm to compare both models. Furthermore, [Huynh *et al.*, 2007] utilized SVM to recognize high-level activities composed of a set of low-level activities or actions. In addition to SVMs, Huynh *et al.* also generated other models such as probabilistic models such as HMMs, clustering models employing the k-means and kNN algorithms. These models were generated in order to compare the performance of the algorithms in the framework that they proposed for AR.

2.1.3.2 Unsupervised Models

Supervised learning is the task of inferring a function from labeled training data. The training data consists of a set of training examples. Supervised learning requires labeled training data that is used by a machine learning algorithm. The algorithm is trained with the labeled data in order to be able to classify new cases. However, unsupervised learning tries to find hidden structures in unlabeled data. It generates a model through density estimations or by pooling similar instances of data.

Although supervised learning has been the principal type of learning employed in AR to date, some researchers have employed unsupervised frameworks due to the problem of labeling the data to train supervised algorithms. Labeling the data is a tedious, difficult and error-prone task. So, a method based on Kohonen Self Organizing Maps is presented in [Krause *et al.*, 2003] and [Clarkson and Pentland, 1999] they use hierarchies of HMM to learn locations and activities such as walking in a supermarket from audio and video data.

The concept of *Eigenspaces* is used in [Huynh and Schiele, 2006] to learn physical activities such as walking and juggling. They employed a multiple eigenspace algorithm, an unsupervised method. However, the same authors utilize the k-means algorithm in [Huynh *et al.*, 2007] and they compared it with other supervised algorithms.

[Patterson *et al.*, 2004b; Liao *et al.*, 2007] employ unsupervised learning based on graphical models. Both works focused on generating models to infer means of transport such as bus, car or walking as well as the destination of the users. [Wyatt *et al.*, 2005] represents activity data as a stream of natural language terms, and activity models were then mapped from such terms. [Minnen *et al.*, 2007] combines data search and exploration with HMM classifiers to discover patterns of short duration movements. Accelerometers were used to capture the data. This work attempts to detect and model body movements to recognize soldier activities in the field.

A very different approach is presented in [Hamid *et al.*, 2005] where activities are represented as groups of *n-grams* (sequences of n consecutive characters) [Shannon, 1948], grouping the classes and characterizing these classes by the frequency of the observed sequences.

2.1.3.3 Semi-supervised Approaches

The semi-supervised learning methods represent a third option. It can be applied when part of the data is labeled and another part, possibly bigger, is unlabeled. Semi-supervised learning is attractive for AR when the cost associated with the labeling is high but it is possible to obtain a small amount of labeled data. There are not many works employing semi-supervised methods until now. [Subramanya *et al.*, 2006] utilizes a dynamic graphical model developed by the authors to recognize the activities that users performed and also the locations where such activities were executed. In [Guan *et al.*, 2007] a new co-training style algorithm in proposed. Furthermore, [Stikic *et al.*, 2008b] analyzed two methods to reduce the amount of labeled data where one of the methods was a semi-supervised algorithm.

2.1.4 Activities

The list of activities that researchers have tried to recognize using sensor networks is very large, which is not surprising since there are many types of applications and sensors. So, this section presents activities that have been used by the AR community in the past.

As it has been shown in section 2.1, in the field of automated planning, plans, goals and activities are words used to define different terms. Thus, activities are the actions that users can execute to change the environment and the objects in it to modify the current state and plans are sequences of actions that users execute to reach a goal. One of the objectives of this thesis is to provide assistance for a user to reach a goal providing a sequence of actions. Most of the activities in the literature, e.g. *cooking* or *making tea*, are the plans that our system will have to find, the cooked recipe is the goal and the sequence of actions that a user has to perform to make tea or to cook will be the plan.

In AR some terms may be considered equivalent to actions and plans, although there is not a generally accepted definition of these terms in the activity-recognition community. So, as we understand them, the term *low-level* activities may be considered equivalent to AP actions and *high-level* activities may be considered equivalent to AP plans. As low-level activities or actions we consider activities such as *picking an object up*, *switching an appliance on* or short duration activities such as *movements* or *gestures* that may compose other activities and are hardly divided in simpler activities. As high-level activities we consider activities that are composed of low-level activities such as *cooking* which may be composed of the actions *peeling potatoes* and *frying potatoes* among many others. Since the plans are composed of a sequence of actions, we consider high-level activities equivalent to plans and low-level activities equivalent to actions.

Most of the activities in the literature are high-level activities. For that reason, from now on we are going to refer to them as just *activities* and we will use the terms *actions* or *low-level activities* to refer to activities that compose other activities or activities with a short duration.

An important type of activities related with health and social care are the activities of daily living (ADL). They were originally proposed in [Katz *et al.*, 1963] and they have became a standard set of activities employed by physicians and caregivers as a measure to estimate the functional status of a person, particularly in relation to people with disabilities and the elderly. The main set of these activities consists of: bathing, dressing, toileting, functional mobility (moving from one place to another while performing activities), bowel and bladder management (recognizing the need to relieve oneself) and feeding.

The set of activities is complemented with other activities in which some objects are involved, as shown in [Lawton and Brody, 1969]. So, this second group of activities is called Instrumental activities of daily living (IADLs) and is composed of: use of telephone or other form of communication, shopping for groceries or clothing, housework, taking medications as prescribed, managing money, using technology (as applicable) and transportation within the community.

The recognition of specific subsets of these groups of activities is shown in [Kasteren *et al.*, 2010a; Philipose *et al.*, 2004; Tapia *et al.*, 2004; Chen *et al.*, 2005; Kasteren *et al.*, 2008]. The recognition of the complete set of activities with sensors is still a challenge since activities such as *managing money* are vaguely defined and others like *toileting* are very difficult to recognize.

Walking, standing or dancing are physical activities that are correctly recognized using inertial or movement sensors since these activities are defined by movements of parts of the body of the user. Information about accelerations and limb positions have been successfully employed for the recognition of these type of activities by several research groups [Huynh *et al.*, 2008; Kern *et al.*, 2003b; Van Laerhoven and Gellersen, 2004; Ravi *et al.*, 2005; Ward *et al.*, 2006; Junker *et al.*, 2008].

In addition to the activities already mentioned, there are more activities that can be recognized with portable sensors. They include martial arts movements [Sun *et al.*, 2002; Chambers *et al.*, 2002; Kunze *et al.*, 2006], cooking [Patterson *et al.*, 2005; Pham and Olivier, 2009], juggling [Huynh and Schiele, 2006], sporting activities such as biking, rowing, running [Choudhury *et al.*, 2008; Ermes *et al.*, 2008; Tapia *et al.*, 2007] or weight training [Chang *et al.*, 2007; Minnen *et al.*, 2006; Minnen *et al.*, 2007] as well as activities in an office environment [Yin *et al.*, 2004; Wojek *et al.*, 2006; Oliver *et al.*, 2002; Begole *et al.*, 2003], carpentry [Lukowicz *et al.*, 2004] and assembling tasks [Ward *et al.*, 2006; Stiefmeier *et al.*, 2006; Stiefmeier *et al.*, 2008]. Activities with a very short duration, also known as gestures, such as using the hand-brake or pushing the brake pedal are explored in [Zinnen *et al.*, 2007; Stiefmeier *et al.*, 2007; Benbasat and Paradiso, 2002].

Besides, part of the works in AR have been dedicated to recognize actions. So, [Amft *et al.*, 2007] split activities in a car assembly scenario into actions, that they call atomic activities. A car body was used to record assembly and testing activities. They gather acceleration data from 47 atomic activities and 11 composite activities. Clarkson *et al.* [Clarkson and Pentland, 1999] present an unsupervised approach for the decomposition of the data provided by on-body camera and microphone. They attempted to discover short duration actions such as *walking into a building* or *crossing the street* and they grouped such actions in high-level activities such as *shopping for groceries* employing HMM hierarchies. [Eagle and Pentland, 2006] utilized location and proximity information through mobile phones in order to detect daily and weekly behavioral patterns. Their work focused on groups of people instead of just one person and they explored things such as the social networks of the users and organizations. In [Amft *et al.*, 2007], authors use a two-stage method to classify arm gestures. Their method consists of localizing and preselecting sections in the continuous signals to later classify the candidate sections selected in the previous stage. Finally, Hoey *et al.* in [Hoey *et al.*, 2011] present a system that recognized the actions that compose the activity of *making tea* mapping the actions directly from the sensors.

2.1.5 Applications

Activity recognition has been applied in a wide range of applications. In the following we outline application areas in which activity recognition has been employed successfully.

One of the main objectives that AR pursues is to enable the creation of new applications related with healthcare. Longer life expectancy is increasing the proportion of the elderly population worldwide. It is hoped that technology advances will help in solving problems like, for instance, helping elderly people to live by themselves longer.

Detecting potentially dangerous situations by detecting vital body signs that indicate imminent health threats is another type of system designed for elderly people [Anliker *et al.*, 2004; Van de Ven *et al.*, 2009; Liszka *et al.*, 2004; Villalba *et al.*, 2006; Chmielewski *et al.*, 2011] as well as detecting when a person has fallen [Wang *et al.*, 2008; Bourke *et al.*, 2007; Jafari *et al.*, 2007]. [Sung *et al.*, 2005; Paradiso *et al.*, 2005] present applications for physical therapy or recovery.

Dementia is a clinical syndrome characterized by the deterioration of a person's cognitive function and memory where the symptoms will gradually get worse. Alzheimer's disease is the most common form of dementia. A need for people with advanced dementia appears from the difficulty they have completing activities of daily living (ADLs) described in 2.1.4. So, [Boger *et al.*, 2005] present a system to help people with dementia capable of monitoring a user attempting a task and offering assistance in the form of task guidance. Audio cues were used in [Boger *et al.*, 2005] to guide the person in performing any missing steps as well as in [Mihailidis *et al.*, 2001] and in [Mynatt *et al.*, 2000] a display can be used to show images of the actions that need to be performed.

Another type of health-related applications promotes a healthier lifestyle. Thus, [Andrew *et al.*, 2007] used wearable sensors and used activity and location information to suggest spontaneous exercises, e.g. to walk to the next bus stop instead of waiting at the current one. [Maitland *et al.*, 2006] estimate and summarize a person's activity levels in order to motivate on daily activities. [Patterson *et al.*, 2004b] propose a system for mentally disabled people that analyzes location information to detect anomalies, e.g. when the user is likely to have taken the wrong bus, and helps the user in correcting the anomalies, e.g. by telling where to get off and which bus to take next.

The prevention of severe medical conditions or diseases before they happen is the objective of other healthcare applications. They employ long-term monitoring to detect changes or unusual patterns in a person's daily life that may indicate early symptoms of diseases such as Alzheimer's. So, [Choudhury *et al.*, 2006] present a system that accumulate and summarize statistics about daily activities and [Van Laerhoven *et al.*, 2004; Paradiso *et al.*, 2005] perform continuous recordings of physiological parameters. These applications can be valuable to estimate the physical well-being of a person but the detection of behavioral changes is still a challenge.

Other types of applications can be found in games and entertainment. The popularity of game controls based on motion sensors, sparked by systems such as Nintendo's Wii platform [Nintendo, 2006] or other devices such as Playstation Move [Sony, 2009] or Kinect [Microsoft, 2009] has made that a wide audience started to use the techniques that AR research communities use since many years ago.

This way, [Medynskiy *et al.*, 2007] presented a wearable RFID system implementing a game interface. [Heinz *et al.*, 2007] used wearable inertial sensors to recognize moves to control martial arts games, [Crampton *et al.*, 2007] created and tested a wearable sensor network that detects body's positions as input for video games, [Zhang and Hartmann, 2007] employed a motion-sensing clamp to control video games and [Ashbrook *et al.*, 2005] present a system to remove the plastic mat in dancing games.

Besides healthcare and entertainment applications, AR has also been employed for industrial applications. These applications can support workers in their tasks and help to avoid mistakes. So, [Amft *et al.*, 2007] presents an architecture that was evaluated in a car assembly scenario using 12 sensor detector nodes to recognize 11 different composite activities. [Stiefmeier *et al.*, 2008] develops a system for tracking activities of workers in car manufacturing plants using information gathered from wearable and environmental sensors. [Ward *et al.*, 2006] combines data from accelerometers and wearable microphones in order to track activities such as sawing or hammering and [Koskimaki *et al.*, 2008] evaluates a system developed for optimization of the steel manufacturing processes. [Lukowicz *et al.*, 2007] investigates the use of wearable computing technology for scenarios in maintenance, production, hospital and fire fighting. In these scenarios, AR and wearable technology were employed for collaborative planning and interaction using wearable devices, integrating and presenting information to assist new workers or context-detection to provide summaries of the performed activities.

AR and wearable systems have been utilized in other areas apart from those already mentioned. Thus, there are applications for dancing [Aylward and Paradiso, 2006; Enke, 2006], sports [Ermes *et al.*, 2008; Minnen *et al.*, 2006], learning of a foreign language vocabulary [Beaudin *et al.*, 2007], categorizing soldier activities [Minnen *et al.*, 2007] or automatic annotation of important events [Kern *et al.*, 2003a; Kern *et al.*, 2007].

2.2 Automated Planning

After the system has been able to recognize the user's actions or activities, through the AR system, some kind of technique is needed to find the next actions the user should perform in order to accomplish his/her goals. In this thesis, automated planning is going to be used to find the sequence of actions for the user to achieve his goal from his current state. The current state will be computed by the AR part of the system and the sequence of actions will be given by an automated planner using a planning domain and the current state. Next, the main concepts about AP will be introduced.

2.2.1 Introduction to Automated Planning

Planning is the process that chooses and organizes the actions required to achieve some desired set of goals by anticipating the action's outcomes. Automated Planning (AP) or AI Planning is an area of Artificial Intelligence that studies this deliberation process [Ghallab *et al.*, 2004]. It is an important component of rational intelligent behavior. In this thesis a system to assist people will be generated. Such system has to be able to, first, recognize what the user has done and, second, provide the user the actions to be performed to achieve his/her goal. Among all the techniques that can be used to solve the second problem, in this thesis we will employ AP.

In this context, three elements can be identified:

1. *Conceptual Model* that describes the elements of a problem solving task.

2. *Representation Language* employed to describe the problems to solve.

3. *Algorithms* that are the techniques used to solve the problems.

There are different forms of planning depending on the characteristics of the conceptual model. So, **Classical Planning** has complete knowledge of world. **Planning Under Uncertainty**, however, works with incomplete models of the world where the outcome of actions might be stochastic and/or there may be incomplete knowledge of the current state. In **Cost-based Planning** actions have an associated cost which is taken into account to obtain plans with minimum cost. Finally, **Temporal Planning** studies how to tackle planning problems when actions might have delayed effects.

These AP subfields focus on solving problems in flat domains. Another subfield of AP, that employs non-flat domains, creates plans by decomposing non-primitive tasks into subtasks, until the decomposition results in primitive tasks which can be directly achieved by executing the primitive actions. This type of planning is called **Hierarchical Planning**.

2.2.2 PDDL

In order to allow for the resolution of problems by a computer, we need to describe the problem in some language. A planning representation language is a notation for the syntax and the semantics of planning tasks. PDDL [Fox and Long, 2003] is the representation language used nowadays by the planning community. It is an attempt to standardize the planning representation languages and facilitate comparative analyses of the diverse planning systems. It was first developed by Drew McDermott and his colleagues in 1998 as the planning input language for the International Planning Competition (IPC)[1], and then evolved with each competition. PDDL includes the STRIPS and ADL representations.

The representation languages such as PDDL separate the model of the planning problem in two major parts:

1. Domain description: definition that describes the state space and the actions that can be executed. The state-space definition contains a definition of object-type hierarchy, a definition of constant objects and a definition of predicates and functions. The actions are described by operator schemas with parameters, preconditions, adds and deletes. Figure 2.2 shows an example of an action in the *Depots* domain. In this

[1]http://ipc.icaps-conference.org/

```
(:action Lift
   :parameters (?x - hoist ?y - crate ?z - surface ?p - place)
   :precondition (and (at ?x ?p) (available ?x)
                      (at ?y ?p) (on ?y ?z) (clear ?y))
   :effect (and (not (at ?y ?p)) (not (available ?x))
                (not (clear ?y)) (not (on ?y ?z))
                (clear ?z) )) (lifting ?x ?y)))
```

Figure 2.2: Example of part of a domain in PDDL.

domain, a set of containers (crates) have to be distributed in warehouses through the use of trucks to move the containers between the different warehouses.

2. Problem description: describes the objects that exist in the problem, the initial configuration of the objects and goals of the problem. Figure 2.3 shows an example of problem definition in the *Depots* domain.

Thus, the domain and problem descriptions form the PDDL-model of a planning task which will be the input of a planner software that will return a plan to solve the planning task.

```
(define (problem depotprob0)
  (:domain Depot)
  (:objects depot0 depot1 - Depot
            truck0 - Truck
            pallet0 pallet1 - Pallet
            crate0 - Crate
            hoist0 hoist1 - Hoist)
  (:init (at pallet0 depot0)
         (at pallet1 depot1)
         (clear crate0) (clear pallet1)
         (at truck0 depot1)
         (at hoist0 depot0) (available hoist0)
         (at hoist1 depot1) (available hoist1)
         (at crate0 depot0)
         (on crate0 pallet0))
  (:goal (and (on crate0 pallet1))))
```

Figure 2.3: Example of a problem in PDDL.

2.2.3 Generating Action Models

This section describes the most relevant works in the literature about the automatic generation of behavioral models (planning domain description). A behavioral model reproduces the required behavior of the analyzed system, in our case a human, such as there is a one-to-one correspondence between the behavior of the original system and the simulated system.

We first describe works that generate models for classical planning. Then, we describe other works that generate other types of models that range from HTN to Partially Observable Markov Decision Process (POMDP).

2.2.3.1 Building classical planning action models

One of the goals of this thesis is to generate a model capable of representing the behavior of a person. To do this, the language PDDL described in section 2.2.2 will be used. Such a model will be built through a planning domain in which the actions of the person will be modelled as planning operators and the goal of the person will be modelled in the planning problem. This way, a planner will assist the person by generating the sequence of actions that the person has to perform to accomplish his/her goal. Next, the most prominent works to learn planning domains in the literature will be presented.

In [Wang, 1995], Xuemei Wang developed a system called OBSERVER that automatically learns planning operators incrementally from plans traces provided by experts. It takes as inputs the predicates, objects and the actions of the domain along with the plan traces. Then, a simulator is used where the experts solve problems and, from the previous and posterior states of each executed action, OBSERVER learns the preconditions and effects of the actions of the domain. The model is refined by observation until the refinement is sufficient to allow planning to take place.

Yang and co-authors presented ARMS in [Yang *et al.*, 2005]. It learns a planning domain from a set of plans consisting of sequences of action names, types, relations, the initial state and the final state of each plan. It generates as output a domain model in the form of STRIPS-type operator schema. The intermediate states of each plan were unknown. Thus, ARMS automatically generated a planning domain, that could be partial, consisting of the operators along with the preconditions, the add list and the delete list of each action.

In [García-Martínez and Borrajo, 1997], its authors presented LOPE, a system that integrated planning, learning and execution. It learned the planning operators from the observations of the effects produced by the execution of the planned actions on the environment.

In the approach presented in [Shahaf and Amir, 2006] action's effects and preconditions are learned in deterministic partially observable domains. It can output expressive operator schema. It requires as input the specifications of fluents, as well as partial observations of intermediate states between action executions.

In [Mourao *et al.*, 2009], its authors presented a technique that learns partially observable planning domains. It only learns the effects of the actions which are the transition rules between states. They built on their previous work [Mourao *et al.*, 2008] where the method only applied to fully observable domains. Their system used deictic coding to generate a compact vector representation of the world state, and learned action effects as a classification problem. [Amir and Chang, 2008] also learned just the effects of actions in deterministic partially observable domains.

In [Lanchas *et al.*, 2007], Lanchas et al. learned the plan-action duration models through regression. They extracted examples from plan executions that are used along with relational regression trees for learning the duration of the actions of a domain.

The LOCM system [Cresswell *et al.*, 2009] automatically induces action schema from sets of example plans. It does not have to be provided with any information about predicates or initial goal or intermediate state descriptions. The example plans are a sound sequence of actions. LOCM exploited the assumption that actions change the state of objects, and require objects to be in a certain state before they can be executed. Planning traces are the input of LOCM, where each action is identified by its name and the objects that are affected or are necessarily present but not affected by the action execution are included.

Opmaker2 [McCluskey *et al.*, 2007; McCluskey *et al.*, 2009] inputs a domain ontology and a solution to a problem, and automatically constructs operator schema and planning heuristics from training sessions. It requires only one example of each operator schema that it learns and an ontology of objects and classes (called a partial domain model) as input. Opmaker2 is an extension of the earlier Opmaker system [McCluskey *et al.*, 2002].

In [Pasula *et al.*, 2007], its authors developed a probabilistic action model representation. They explored the learning of relational rule representations in stochastic domains. In the domain of first-order logic they learn rules that given a context and an action provide a distribution over results.

In [Jiménez *et al.*, 2013], the PELA architecture is presented. The architecture is based on the integration of a relational learning component and the traditional planning and execution monitoring components. The learning component allows PELA to learn probabilistic rules of the success of actions from the execution of plans and to automatically upgrade the planning model with these rules. It automatically upgrades the deterministic domain that is used at the beginning as it learns knowledge about the execution of actions. The upgrade consists of enriching the initial STRIPS action model with estimates of the probability of success of actions and predictions of execution dead-ends. The upgraded models are used to plan in probabilistic domains.

2.2.3.2 Building other types of action models

Besides classic planning domains, HTN models can be used to represent the behavior of the users. Due to their effectiveness in real problems, HTN's may obtain better results for the system to assist the user.

It is possible to generate a classical planning domain from sensors' readings employing the techniques presented in the previous section 2.2.3.1. The flat classical domain can be used to generate hierarchical domains for HTN. In [Reddy and Tadepalli, 1997], its authors used inductive generalization to learn task decomposition constructs, which relate goals, subgoals, and conditions for applying goal-decomposition rules (d-rules). By grouping goals in this way, the learned task models solve problems faster. Other works that learn HTN models from plans and an action model are [Choi and Langley, 2005; Ruby and Kibler, 1991].

[Nejati *et al.*, 2006] describes an approach that observes sequences of operators taken from expert solutions to problems and learns hierarchical task networks from them. The authors describe how they induce what they called "teleoreactive logic programs" that index methods by the goals they achieve.

[Hogg *et al.*, 2008] presents the system called HTN-MAKER which is capable of generating HTN domains from classical planning domains, a collection of plans and a set of task definitions for the composed operators and generates a HTN domain model. The authors of ARMS and HTN-MAKER in [Zhuo *et al.*, 2009] develop the HTN-learner algorithm that builds constraints from given observed task decomposition trees to build action models and method preconditions. Then, the constraints are solved employing a weighted MAX-SAT solver. It does not depend on complete action models or state information.

Other different approaches utilize POMDP to generate plans instead of operators. Thus, the authors of [Holmes and Isbell Jr, 2005] employ schema learning to discover probabilistic action rules using discrete sensors. The system observed the action effects on the environment and predicted the effects that such actions produced. It built what they called schemas. A schema $C \xrightarrow{a_i} R$ indicates that when the action a_i is employed in the situation C, the result R is produced.

Finally, in [Hoey *et al.*, 2011], its authors presented a system capable of learning a POMDP also from sensor readings. The system is created mapping sensor readings directly into actions. For instance, each time the sensor s is activated, the system detects that action a_s is executed. Also, they used what they called “virtual sensor” in order to recognize one of the actions using an accelerometer and employing a machine learning method for recognizing such action. This way, the system created a mini AR system just for one action. Once they have mapped the actions and the states, they build a POMDP. So, the entire system recognizes the user actions and maps both belief states and action observations into choices of actions. Hence, they build the POMDP that is used to monitor a person’s progress in a task and it prompts the users whenever they get stuck in their activities. Our overall objective is similar to this approach but we will generate action models based on AP domains instead of a POMDP. A POMDP does not scale well in general. Also, domains written in PDDL are easily readable and maintained by users. Finally, using PDDL models allows us to build on top of current state of the art powerful domain-independent planners.

Part II

Objectives and Evaluation

Chapter 3

Objectives of the Thesis

This chapter describes the goals of this Thesis. The overall objective of this thesis is to develop techniques for activity and action recognition in order to be used for the generation of planning domains. The generated planning domains should model the behavior of the users of the system in order to build an assistant for them. Such assistant has to be capable of proposing the users the next action they have to perform to accomplish their goals whilst they try to complete their activities by themselves in a smart environment. To that end, a sensor network has to be built and installed in a domestic environment to gather information about the actions that the user performs.

The specific objectives of this doctoral thesis can be detailed as:

1. To define and develop a sensor network for capturing the effects of users actions. Such sensor network will be composed of the hardware (physical sensors, cables, batteries, etc) and the software to control the sensors. The hardware will be composed of a set of sensors capable of gathering relevant information to recognize users actions in the domestic environment while he/she tries to accomplish his/her goal. Besides, the employed hardware must permit users to perform their actions correctly without interfering with them. The software will process the signals provided by the sensors, filter them, correct them and recover incorrect data whenever it is possible; finally, it will synchronize the data from the whole set of sensors employed. Thus, it will provide the necessary information in a correct format to be utilized to learn the actions.

2. To develop a learning system that integrates the complete cycle of inference of the performed actions. The learning system will use as input the information provided by the sensor network described in the previous objective.

3. To define new methods for the analysis of temporal series as well as the inclusion of new attributes on different activity recognition algorithms. These methods should be able to help on the later steps of learning planning domain models. So, they should focus on the recognition of the start and end of the activities.

4. To evaluate the performance of the new methods on data captured in a real environment where the previously defined sensor network is installed.

5. To develop computational models for the automatic generation of a planning model capable of reproducing user behavior, using the information provided by the sensor network.

6. To develop computational models to predict the possible actions that the user will perform using the planning domains generated in the previous objective.

Chapter 4

Evaluation

To evaluate the methods that will be developed in this thesis, we will compare the performance obtained by our methods with other algorithms existing in the literature when possible. Since this thesis has two parts related with two different research areas, activity recognition and automated planning, we will use the criteria that the scientific community employs for the evaluation of each discipline.

4.1 Evaluation of the Activity Recognition Algorithm

For the evaluation of the performance of the AR methods, we will use the criteria employed by the AR community. They usually employ two metrics: *precision* and *recall*. They are computed as follows:

$$\text{Precision}=\tfrac{1}{C}\textstyle\sum_{c=1}^{C}\left\{\frac{tp_c}{tp_c+fp_c}\right\}$$

$$\text{Recall}=\tfrac{1}{C}\textstyle\sum_{c=1}^{C}\left\{\frac{tp_c}{tp_c+fn_c}\right\}$$

where tp_c are the true positives of class c, fp_c are the false positives of class c, fn_c are the false negatives of class c and C is the number of classes. Precision is the fraction of the whole instances that are correctly classified, while recall is the fraction of the instances belonging to one class that are correctly classified.

These metrics are used very often in the literature [Kasteren *et al.*, 2010c; Chawla, 2010]. However, there are no standard metrics to evaluate AR systems and there are some publications where these metrics are not used.

With regards to the datasets to be employed to validate the methods, again there are no defined standards. The community has not specified a minimum number of tests to be executed nor a number of persons to perform the activities or plans to be validated. Most of the literature utilizes datasets specifically built by the researchers to test the addressed problem in the publication and, with a few exceptions, the datasets employed are not made public for the rest of the community. Therefore, whenever possible, we will utilize public datasets captured by other researchers to validate the developed techniques.

Two datasets will also be generated in this thesis for the validation in cases where there are no public ones with the needed characteristics. The first dataset will be captured employing a simulator in order to imitate the behavior of a sensor network. A second dataset will be gathered using some sensors, given that none of the previously published datasets is adequate for the validation of our work.

So, in [Ortiz *et al.*, 2008] we report on a dataset using a simulator to validate the proposed system. In [Ortiz *et al.*, 2011] we employed public datasets. Such datasets are described in [Kasteren *et al.*, 2008; Patterson *et al.*, 2005]. In Chapter 7 we employed the public datasets described in [Huynh *et al.*, 2008; Kasteren *et al.*, 2008; Patterson *et al.*, 2005]. Finally, in Chapter 8 we generated a new dataset.

4.2 Datasets

This section describes the datasets we have used to evaluate this thesis.

4.2.1 Publicly Available Datasets

4.2.1.1 Kasteren Dataset

This dataset was recorded and used by Kasteren and colleagues in [Kasteren *et al.*, 2008]. The sensor network consists of wireless network nodes to which simple off-the-shelf sensors can be attached. Each sensor sends an event when the state of the digital input changes or when some threshold of the analog input is reached. The dataset was recorded in the house of a 26-year-old man living alone in a three-room apartment where 14 state-change sensors were installed. Locations of sensors included doors, cupboards, refrigerator and a toilet flush sensor. Sensors were left unattended, collecting data for 28 days in the apartment. The dataset contains 2638 sensor events and 245 activity instances. Activities were annotated by the subject himself using a bluetooth headset as described in [Kasteren *et al.*, 2008]. Seven different activities were annotated, namely: *Leave house*, *Toileting*, *Showering*, *Sleeping*, *Preparing breakfast*, *Preparing dinner* and *Preparing a beverage*. Times where no activity is annotated are referred to as *Idle*. The dataset is public and can be downloaded with its annotations from `https://sites.google.com/site/tim0306/datasets`.

4.2.1.2 Patterson Dataset

This dataset was used and described in [Patterson *et al.*, 2005]. The experiments performed with this dataset focused on routine morning activities which used common objects and are normally interleaved. The 11 activities which were observed are: *Using the bathroom*, *Making oatmeal*, *Making soft-boiled eggs*, *Preparing orange juice*, *Making coffee*, *Making tea*, *Making or answering a phone call*, *Taking out the trash*, *Setting the table*, *Eating breakfast* and *Clearing the table*. To create the dataset, one of the authors performed each activity 12 times in two contexts: twice in isolation, and then on 10 mornings all of the activities were performed together in a variety of patterns.

In order to capture the identity of the objects being manipulated, the kitchen was outfitted with 60 RFID tags placed on every object touched by the user during a practice trial. Data is in the form: <objectID> <activityID>. This dataset is more challenging than the previous one; most tasks were interleaved with or interrupted by others during the 10 full data collection sessions. In addition, the activities performed shared objects in common. This made interleaved AR much more difficult than associating a characteristic object with an activity. The dataset is public and can be downloaded from http://www.cs.rochester.edu/u/kautz/Courses/577autumn2007/a5data.zip.

4.2.1.3 Huynh Dataset

This dataset was recorded and used by Huynh and colleagues in [Huynh *et al.*, 2008]. They recorded 34 activities using two sensors with a 3D accelerometer. The sensors were worn on a wristband and into the subject's pocket to record the subject's daily life over a period of sixteen days. In total, this dataset consists of 164 hours of recordings. Of these, they had to discard 28 hours due to failures in the sensor hardware. The activity set consist of the following activities, along with the unlabeled class: *sitting / desk activities*, *lying while reading / using computer*, *having dinner*, *walking freely*, *driving car*, *having lunch*, *discussing at whiteboard*, *attending a presentation*, *driving bike*, *watching a movie*, *standing / talking on phone*, *walking while carrying something*, *walking*, *picking up cafeteria food*, *sitting / having a coffee*, *queuing in line*, *personal hygiene*, *using the toilet*, *fanning barbecue*, *washing dishes*, *kneeling / doing sth. else*, *sitting / talking on phone*, *kneeling / making fire for barbecue*, *setting the table*, *standing / having a coffee*, *preparing food*, *having breakfast*, *brushing teeth*, *standing / using the toilet*, *standing / talking*, *washing hands*, *making coffee*, *running*, and *wiping the whiteboard*.

This dataset does not provide the raw data; instead, it provides subsampled data containing the mean and variance of each accelerometer over a sliding window of 0.4 seconds of the raw data. That way they reduce the sampled rate from 100Hz to 2.5Hz. The dataset is public and can be downloaded from http://www.d2.mpi-inf.mpg.de/datasets.

4.2.2 Datasets Generated in this Thesis

In this section we present the datasets we have captured in order to test some of the methods developed in this thesis.

4.2.2.1 Simulator Dataset

A dataset was generated for testing the first learning algorithm for AR that we will describe in section 5. The simulator is composed of a server, a client and a compiler. The server simulates a world in two dimensions, defined declaratively in a file that contains the configuration of the simulated home. The information that the file contains is: the home size, layout, which objects it contains, where they are, which tags there are, what readers it contains, and where they are. Once the server is running, any client can connect to it and issue commands. Currently, those commands are: *move*, *pick-up* objects or *put-down* objects. The server simulates the environment and generates a log with the sensor readings produced by the user's interaction with the environment. This can be done in two ways: logging the sensor periodically; or recording the sensor readings when they change their value (event-driven).

On the other end, there is a client to simulate a user performing activities of daily living. It performs some preprogrammed high-level actions or plans composed of low-level actions or primitive tasks. The client randomly chooses the durations of those actions. The order of execution of some low-level actions is also changed randomly. For instance, in the high-level action of “brushing teeth” the client can pick up first either the toothpaste or the toothbrush.

In this dataset we have chosen some high-level actions that need to use objects placed in the same area, to provide more sensor readings with no low-level actions associated, thus obtaining more false positives. A total of eight high-level activities have been selected: *brushing teeth*, *combing hair*, *shaving*, *throwing out the safety razor*, *throwing out the toothpaste*, *using the vacuum cleaner*, *ironing* and *sweeping*. For instance, both brushing teeth and throwing out the toothpaste are defined in terms of the low-level action “pick-up toothpaste”.

The dataset is composed of traces of a client connected to the server. In order to provide more realistic data, such client was programmed with behaviors taken from real persons observed in a flat. Those persons were performing some daily activities at home and all the low-level actions the persons performed were annotated. The actions executions observed from humans behavior did not take always the same time. So, given that we generated random example traces that modelled how those human subjects behaved, we set the range of potential values of the actions duration by observing how long it took those actions in the real world. Also, some activities were performed always in the same order, so we changed the order of some low-level actions randomly in order to obtain more variety.

The server can simulate both false negatives and sensor failures. So, the server was executed with a 0%, a 5% and a 10% level of each kind of noise (false negatives and sensor failures). False positives are present in all cases. The client has been executed during four simulation days. All high-level actions were performed in different ways every day and some of them were performed more than once in the same day. A total of 12 activities and 40 actions were performed. The simulated person executed eight different plans to accomplish eight kinds of goals (represented by the previously mentioned eight high-level activities). An example of a plan could be: *move to the bathroom*, *pick-up the toothbrush*, *pick-up the toothpaste*, *put-down the toothpaste* and *put-down the toothbrush after a while*.

From the execution of those plans we obtained between 278 instances in the case of no noise, to 410 instances in the case of 10% noise levels.

4.2.2.2 Kitchen Dataset

In order to test the whole system, a dataset has been generated with the task of making an omelette and the actions that compose this activity. The actions are *open*, *close*, *get-out*, *put-away*, *pick-up*, *put-down*, *crack-egg*, *transfer*, *switch-on*, *switch-off*, *fry*, *beat* and *null*. *null* is assigned to the events not involved in any action. Activities performed in a kitchen have been studied and included in many works in the past like in [Hoey *et al.*, 2011; Patterson *et al.*, 2005] and some of them are in public datasets like [Patterson *et al.*, 2005]. However, none of the public datasets is suitable to test our complete system, since they do not provide enough information to classify the actions that compose the activities stored in the datasets. For instance, in the case of [Patterson *et al.*, 2005], the data is labeled with the name of the activities; e.g. *Making Coffee* or *Setting the table*; but not the labels of the actions that compose those activities; e.g. *switch on the coffee maker* or *put down a plate*; which is what we need. For that reason we have designed a sensor network focused on getting that information.

The sensor network employed to record the data is composed of several types of sensors. We have used magnetic sensors, RFID sensors, and cameras. Magnetic sensors have been used to monitor the state (opened or closed) of the cupboards, drawers, and the fridge of the kitchen. Two RFID receivers are used to track the objects that the user holds in both hands through two RFID gloves similar to the ones used in [Medynskiy *et al.*, 2007]. The RFID readers employed operate at 125 kHz frequency and the antenna attached to the glove detects tags within 2–10 centimeters of its center. 125 KHz tags are not affected by water or metals, what makes them very appropriate for the environment in which we used them. Finally, we used four cameras to detect the state and location of the objects in the kitchen. The objects used are: a bowl, forks, plates, a fry-pan, an oil bottle, and a salter. All of them have RFID tags attached to be detected. The ingredients are eggs, salt, and oil. Also, there is a cooktop as appliance, the kitchen top as working surface, and a sink.

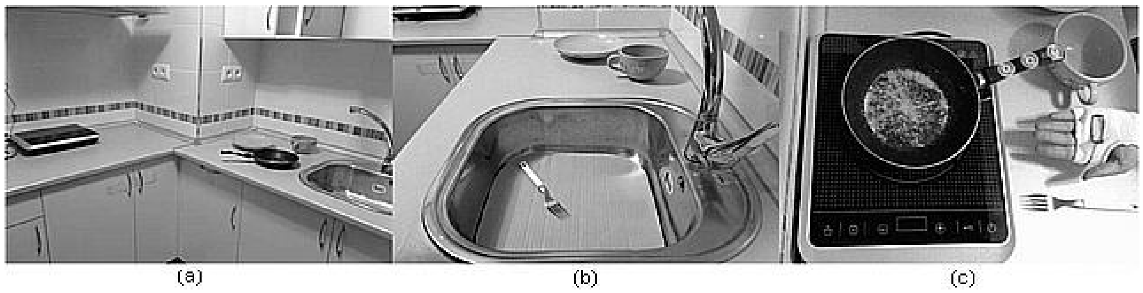

Figure 4.1: (a) kitchen; (b) sink and part of the kitchen top with some objects (c) cooktop, glove with the RFID reader and some objects with RFID tags.

Figure 4.1 shows three pictures. The first one in Figure 4.1(a) shows the kitchen where the user cooks. Figure 4.1(b) shows the view of one of the cameras. In this case, the camera focused on the sink and part of the kitchen top where some objects have been placed. Finally, Figure 4.1(c) shows the view of another camera focused on the cooktop. Also, this picture shows one of the RFID gloves used by the user and some objects with RFID tags. A third camera is focused the same place than 4.1(c) but from a different angle. The fourth camera is placed in the left edge of the kitchen top where the cooktop is.

We used the *OpenCV* library [Bradski, 2000] to recognize the objects. First, our system removes the background using an initial photo. Then, part of the shadows of the objects is removed using the initial photo but darkened. Finally, color and shapes are used as features to recognize the objects and their states using a classifier. Each object has a different color to facilitate the recognition. Before recording the dataset, the computer vision system was trained taking photos of all the objects of the kitchen in many situations. In each photo, the background is removed and the color and shape are extracted using contours. This way, a classifier was generated. Some of the algorithms provided by *OpenCV* were tested: *Support Vector Machines* (SVM) [Platt, 1999], *k-nearest neighbor* (kNN) [Aha *et al.*, 1991] and *Random Forests* (RF) [Breiman, 2001]. The best results were achieved using SVM, which obtained a recognition rate of 98% of the objects. Inside some objects ingredients can be found in different states: oil in the fry-pan, raw egg in the bowl, beaten egg in the bowl, beaten egg in the fry-pan, omelette in the fry-pan and omelette on the plate. Each ingredient in each state in each object has been recognized as if they were different objects. The recognition rates were 63% for the fry-pan classifier, and 71% for the bowl classifier. The best algorithms were employed to recognize objects and situations in order to generate the logs that were part of the dataset.

In order to avoid problems caused by occlusion, there is more than one camera focusing on the working surface. Also, no changes are reported by the cameras until detecting an object and its state. So, if the user hides an object during some seconds, the cameras will not report any information about that object, and the system will consider that the object is in the same location and state previous to be hidden. Cameras work together with the RFID's, so, whenever an object is detected by an RFID, the cameras try to find that object. The system also searches for specific information in specific places (e.g. the eggs just can be fried in the fry pan or beaten in the bowl). To determine the state of the appliances we use the area of the images where a display indicates the state of the appliance. Then, using the difference of the colors in that area, the state is computed. Finally, the initial state of some elements is defined since the system does not have sensors to detect such information. For instance, objects that are initially inside some cabinet or inside the fridge. In addition, we have attached some RFID tags to the cabinets, drawer, fridge and surfaces in order to have some information about the place where the user is.

The task of preparing the omelette has been performed by two different users ten times each, from beginning to end. The task was performed by two users to avoid the possible bias caused by a single user. The task was performed ten times by each user in order to obtain enough instances of each action to test the models. Also, parts of the task have been recorded in order to get more instances of some of the actions that compose the task. For instance, beating the eggs and putting the mix in a fry-pan or picking an egg up from the fridge and cracking it.

The sensor network used has some limitations. The user must use RFID gloves in order to recognize the object that he (she) picks up. Also, different objects cannot have similar colors. Both limitations could be solved using accelerometers like in [Amft *et al.*, 2007; Hoey *et al.*, 2011] to detect the object that is being used. That way, the system could infer the actions performed using each object as well as its locations and state.

4.3 Evaluation of the Algorithm to Generate Planning Domains

To generate planning domains, we need a dataset that includes information about the high-level activities (e.g. cooking, cleaning) carried out by the user and the low-level activities (e.g. picking fry-pan up, closing cupboard) that compose high-level activities. After analyzing most of the publicly available datasets, none of them was found suitable for our needs. Consequently, for the evaluation of this part of the thesis a new dataset has been generated. The description of the dataset captured for this purpose can be found in Section 4.2.2.2 named *Kitchen Dataset*.

Our AR system is capable of recognizing the low-level activities that users perform while they cook; that is, it recognizes the actions that compose the high-level activity "cooking an omelette". Once the actions are known, the system will be able to propose the next action or actions to be performed by the user to complete the recipe through the plan or plans generated by a planner. The planner will employ the planning domain automatically generated from the sensorial information and the actions recognized by the AR system in the previous phases. The dataset generated, "cooking an omelette", permits us to evaluate the generated planning domains and also the performance of the overall system.

The standard method to evaluate the automatically generated planning models is to create a number of problems using a generator in several domains. Then, the problems are solved using the automatically generated planning domain and a baseline planning domain.

Finally, the numbers of solved problems are compared. This is the approach used in [Yang *et al.*, 2005; Mourao *et al.*, 2009]. In our domain, we can not use a generator to create problems and the only available problems are those in the datasets. So, we select part of the dataset to generate the domains and the other part to generate planning problems to test the domains. The planning problems are generated selecting randomly states present in the part of the dataset employed for testing as initial and goal states.

4.4 Evaluation of the Generated Domains

After generating the planning domains, these are used to predict and recognize actions. In order to evaluate the predictions and the recognition of the actions using the generated domains, part of the dataset is used to generate the domains and the other part to evaluate the predictions.

To evaluate the predictions of the generated domains, from every state of the part of the dataset employed for testing, a planning problem is generated where the initial state is the current state and the goal state is to have the omelette cooked. Then, the planner is executed to solve the problem and the first action returned by the planner is compared with the action that the user executes after the current state.

To evaluate how the generated domains recognize actions, from every state of the part of the dataset employed for testing, a planning problem is generated where the initial goal is the current state and the initial state is the state before the current one. Then, the planner is executed to solve the problem and the first action returned by the planner is compared with the action that user executes between both states.

The main difference between both tests is that the action has already occurred to recognize actions. Then, the last states are used and the action between those states is used for the evaluation. On the other hand, to predict an action, just one state is needed since the goal state is known (to have the omelette cooked) and the state after the action and the action are not known since they have not occurred yet.

Part III

Methods

Chapter 5

Relational Learning Algorithm for Activity Recognition

This chapter presents an algorithm to infer the actions that people perform in order to accomplish activities of daily living starting from sensory inputs. The approach is based on using relational learning to infer predictions about which action has just been executed. A model is learned for recognizing executed actions based on the state changes detected from sensor readings. The experiments were executed using an environment simulator feeded by data gathered from real human behavior [Ortiz *et al.*, 2008].

The principal motivation for the development of this algorithm was to use relational learning for recognizing the actions. Relational learning uses a representation based on predicates and AP also uses a predicate logic representation. This way, once the information from the sensors has been represented using predicate logic, it can be used for AR and the automatic generation of a planning domain.

5.1 Introduction

Understanding activities performed by humans has been recognized as a capability with a wide range of applications. Those applications include tracking activities [Fishkin *et al.*, 2005; Bao and Intille, 2004; Tapia *et al.*, 2004], behavioral monitoring [Mihailidis *et al.*, 2001], prompts to help in completing activities [Boger *et al.*, 2005], detection of failures [Lindner *et al.*, 2005] or surveillance [Niu *et al.*, 2004]. Many systems address the problem of activity recognition from different points of view. Some approaches use sensors placed in the body (e.g. [Bao and Intille, 2004; Kahn *et al.*, 1999; Kasten and Langheinrich, 2001]), sensors installed in home environments [Tapia *et al.*, 2004; Fishkin *et al.*, 2004] and other approaches use both, sensors installed in home environments and sensors placed in the body of the user [Logan *et al.*, 2007].

In this work, this last approach is used in order to obtain as much information as possible from the sensors. Within this work, a software simulator of a real home has been built. In such system, different kinds of sensor devices (RFID, Infrared, etc) can be modelled. They will provide information about the objects that the user interacts with. The system logs the inputs of the sensors and analyzes them in order to obtain a model of user behavior. Although any number of sensors can be used and placed anywhere, in this work we used a simulation of RFID readers placed in the user, like the ibracelet in [Fishkin *et al.*, 2005]. Those readers can detect RFID tags up to 10 cm. Also, all "interesting" objects are tagged with passive tags. The simulator generates noise as false negatives, sensor failures and false positives in order to provide realism. The false negatives are produced when a tagged object is picked up and the reader cannot detect it. The sensor failures are generated when the reader is detecting a tag, and then the reader fails changing the signal detected to inactive. The false positives are yielded by tags placed close to the reader that the user does not grab.

The overall aim of this thesis is to apply task planning [Ghallab *et al.*, 2004] to recognize actions, plans, and goals. To achieve this goal, we are interested as a first step in learning first-order logic action models from sensor readings and recognize actions. Some actions can be mapped directly from sensor readings depending on the type of sensor that the system uses, but others have to be learned because the information that the sensor provides is not enough. Therefore, our goal in this work is to build a system (composed of a set of sensors, a compiler and a machine learning system) to learn the actions that users perform in order to carry out activities of daily living (ADLs). The employed sensors are capable of sensing which tagged objects are closer than 10 centimeters, but they are not able, for instance, to sense moving tags. Therefore, the challenge of this work is to detect when an object has been picked up or put down to perform a high-level action and during how long those objects are typically held.

5.2 System's Architecture

The system is composed of a server, a client and a compiler. The server simulates a world in two dimensions, defined declaratively in a file that contains the configuration of the simulated home. The information that the file contains is: the home size, layout, which objects it contains, where they are, which tags there are, what readers it contains, and where they are. Figure 5.1 shows the environment simulated by the server where the sensors are indicated with small black circles. Once the server is running, any client can connect to it and issue commands. Currently, those commands are: *move*, *pick-up* objects or *put-down* objects. The server simulates the environment and generates a log with the sensor readings produced by the user's interaction with the environment. This can be done in two ways: logging the sensor periodically; or recording the sensor readings when they change their value (event-driven).

We have also developed a client to simulate a user performing activities of daily living. It performs some preprogrammed high-level actions or plans composed of low-level actions or primitive tasks. The client randomly chooses the durations of those actions. The order of execution of some low-level actions is also changed randomly. For instance, in the high-level action of *brushing teeth* the client can pick up first either the toothpaste or the toothbrush.

Once the server has generated a log, we use the compiler to detect the state changes of the sensors and to generate a learning file that contains instances to be used by the learning component. Independently of the way the server uses to produce the log, the compiler generates always the same learning file. Figure 5.2 shows the architecture of the system.

We use the ACE Data-Mining System as the machine-learning component [Blockeel *et al.*, 2006]. It is a machine-learning tool that provides a common interface to a number of relational data mining algorithms. Among others, ACE includes Tilde [Blockeel and De Raedt, 1998], that we have used in our experiments. Tilde builds relational decision trees from relational instances.

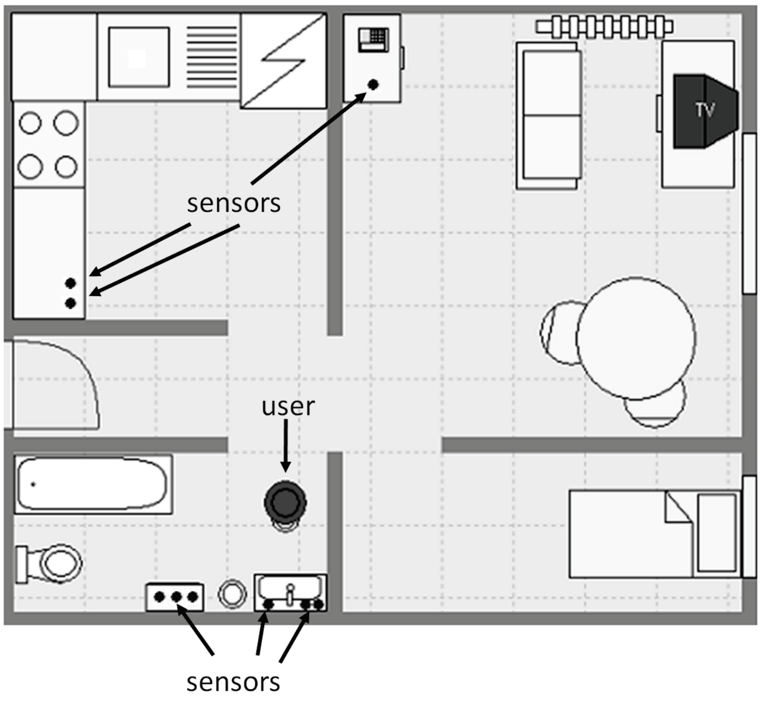

Figure 5.1: Simulated house.

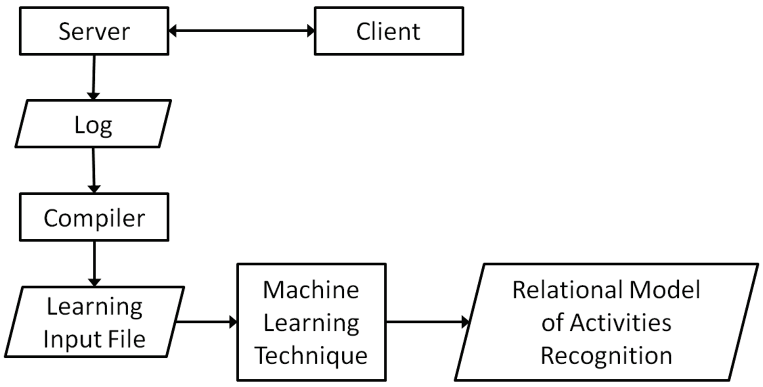
Server
Client
Log
Compiler
Learning Input File
Machine Learning Technique
Relational Model of Activities Recognition

Figure 5.2: High level view of the system architecture.

5.3 Generation of Learning Instances

From the log that the server generates, state changes have to be mapped into low-level actions. We define *state change* as any change produced in the state of any sensor. In order to carry out this task, first the log has to be filtered in order to detect these changes. This is the task of the compiler. It reads the server log and generates learning instances. In the log, each reading includes information about the time, date and the sensors readings. If the reader detects no tags, it logs the "inactive" signal. Figure 5.3 shows an example of the log. In the example, the first reading means that the user reader does not detect any tag. The second reading means that the user reader started to detect the toothpaste and the toothbrush.

The predicate used in the logs is:

```
06:15:15 15.02.08
sensor(readerUser_user1,inactive,0)

06:15:16 15.02.08
sensor(readerUser_user1,toothbrush,0.0)
sensor(readerUser_user1,toothpaste,0.0)

06:15:16 15.02.08
sensor(readerUser_user1,toothbrush,0.131)
sensor(readerUser_user1,toothpaste,0.131)

...

18:50:31 16.02.08
sensor(userReader_usuario1,comb,0.0)
sensor(userReader_usuario1,shavingfoam,3.105)
```

Figure 5.3: Example of a log.

sensor(Id,Reading,Duration): information on the sensor at that time step. The arguments relate to the sensor id, the actual reading, and how long the sensor has been activated

Algorithm 1 shows how the AR model is built. Actions are modelled in terms of two states (before and after) involved in each action, so the compiler searches for different consecutive readings to detect state changes. Function $readLog(log, counter)$ returns the reading of the log indicated by $counter$. When the compiler detects a state change in the sensor readings (done by the function $isStateChange(stateAfter)$), it creates a learning instance using the two states (before and after) involved. RFIDs provide information about the objects that start and stop being detected. When an object starts being detected ($detectedNewObject(stateAfter)$), the system searches in the log ($searchDuration(log, stateAfter)$) to find when the object stops being detected in order to detect how long the object has been detected. This way, when an object starts being detected, the created instance has the amount of time ($duration$) that the object is going to be held. The sensors are not able to distinguish between picked up objects and objects close to the hand. This information, the duration, is used to help the system to differentiate these two situations. Each reading in the log has the action that is being performed. Thus, when the instances are created ($featureExtraction(stateAfter, stateBefore)$), $stateAfter$ contains the class of the instance that is going to be created.

The instances are composed of the predicates *sensorBefore(X,Y,Z)* and *sensorAfter(X,Y,Z)* where X is the identifier of the instance, Y is the identifier of the objects detected by the RFIDs, and Z is the time that the object is detected. The predicate *sensorBefore* is used to indicate the objects that are detected before the state change and *sensorAfter* is used to indicate the objects that are detected after the state change. More than one object can be detected at the same time so, in each instance there can be more than one of both predicates. Every time a state change is produced, the simulator generates a learning instance that automatically labels. The predicate *action(X,C)* is used to label each instance indicating the class in Z.

Figure 5.4 shows an example of three relational learning instances. Instance `ej6` shows that the user1 sensor detects the toothbrush after the state change, and it is detected during 40 seconds. Also, the toothpaste is going to be detected during five seconds. Both objects have been *picked up*. Instance `ej15` shows when the user *puts down* the toothpaste that

Algorithm 1 Model creation algorithm.

Input: log

Output: learning model

```
counter ← 0
instances ← *empty*
stateAfter ← *empty*
stateBefore ← *empty*
stateAfter ← readLog (log, counter)
counter ← $counter + 1$
while stateAfter $\neq$ *null* do
  if (isStateChange (stateAfter)) then
    if (detectedNewObject (stateAfter)) then
      duration ← searchDuration (log, stateAfter)
      stateAfter ← update(stateAfter)
    end if
    instance ← featureExtraction (stateAfter, stateBefore)
    instances ← instances + instance
  end if
  stateAfter ← readLog (log, counter)
  counter ← $counter + 1$
  stateBefore ← stateAfter
end while
mlModel ← TILDE (instances)
return mlModel
```

was detected in instance `ej6`. And, instance `ej22` shows an example of a false positive; the user1 sensor detects the toothbrush just during 0.1 seconds. Such instance is generated when the user *moves* next to the toothbrush and the RFID detects it for an instant. In that moment the user is holding a razor in the other hand.

```
action(ej6,pickup).
sensorBefore(ej6,inactive,0).
sensorAfter(ej6,toothbrush,40).
sensorAfter(ej6,toothpaste,5).
...
action(ej15,putdown).
sensorBefore(ej15,toothbrush,40).
sensorAfter(ej15,inactive,0).
...
action(ej22,move).
sensorBefore(ej22,razor,7.721).
sensorAfter(ej22,razor,7.721).
sensorAfter(ej22,toothbrush,0.1).
```

Figure 5.4: Three training instances generated by the compiler.

5.4 Learning Action Models

Once the compiler generates all training instances using the events, we use Tilde to learn a model of the relation between actions and states. Generating high-level representations from low-level data, as sensor readings, is a well-known problem in AI, as shown, for instance, in robotics [Fox *et al.*, 2006]. In our case, we have defined seven extra predicates to perform this step. The predicates `increase` and `decrease` tell if, in the state change, the number of objects detected by the user reader increases or decreases with respect to the previous reading. When an RFID changes its state from no detecting anything to detect an object, Tilde could use that information since the state changes from "inactive" to "objectDetected". But, when a second object is detected by the sensors, a second predicate is created. Without the predicate `increase` Tilde could not use this extra information. The predicate `decrease` helps when the objects detected decrease but still there are objects detected. In addition, the predicates `durationLessAfter, durationLessBefore, durationMoreAfter` and `durationMoreBefore` define whether the object has been detected longer than a given period of time in the state before and in the state after the instance. The period of time is automatically found by Tilde. Each predicate is defined as a Prolog rule as Tilde allows background knowledge to be specified intentionally. Figure 5.5 shows the corresponding rules.

```
length([],0):-!.
length([inactive|B],N):- !, length(B,N).
length([A|B],N):- length(B,N1), N is N1+1.

objectsBefore(A,N):- findall(X,sensorBefore(A,X,_),Bag),length(Bag,N).
objectsAfter(A,N):- findall(X,sensorAfter(A,X,_),Bag),length(Bag,N).

increase(A):- objectsBefore(A,N1), objectsAfter(A,N2), N1<N2.
decrease(A):- objectsBefore(A,N1), objectsAfter(A,N2), N1>N2.

durationLessAfter(A,D):- sensorAfter(A,_,C),C<D.
durationLessBefore(A,D):-sensorBefore(A,_,C),C<D
durationMoreAfter(A,D):- sensorAfter(A,_,C),C>D.
durationMoreBefore(A,D):-sensorBefore(A,_,C),C>D
```

Figure 5.5: Background knowledge used for learning actions models.

As with respect to the target concept, we are trying to infer the low-level action that the user has applied from the two consecutive perceived states (before and after). We are using the three low-level actions: *move*, *pick-up* and *put-down*. Taking as input the training instances, the background knowledge and the target concept, Tilde builds a relational decision tree. Using instances like those in Figure 5.4 and the predicates in Figure 5.5, Tilde generates a tree like the one in Figure 5.6.

Using the classification tree in Figure 5.6 every future example can be classified by following the corresponding tree branch. For instance, first, it checks if `decrease(A)` is true. If so, it checks whether `durationLessBefore(A,0.1)` is true, and then whether the predicate `durationMoreBefore(A,2)` holds. If it is false, it predicts the action performed to be `move`. Otherwise, it checks whether `durationMoreBefore(A,0.5)` holds, and so on.

```
action(-A,-B)
decrease(A) ?
+--yes: durationLessBefore(A,0.1) ?
|   +--yes: durationMoreBefore(A,2) ?
|   |     +--yes: [move]
|   |     +--no:  durationMoreBefore(A,0.5) ?
|   |           +--yes: [pickup]
|   |           +--no:  [move]
|   +--no:  durationLessAfter(A,0.1) ?
|       +--yes: durationLessBefore(A,4) ?
|       |    +--yes: durationLessBefore(A,1) ?
|       |    |    +--yes: [putdown]
|       |    |    +--no:  durationLessBefore(A,3) ?
|       |    |        +--yes: durationLessBefore(A,2)?
|       |    |        |     +--yes: [putdown]
|       |    |        |     +--no:  [putdown]
|       |    |        +--no:  [putdown]
|       |    +--no:  [putdown]
|       +--no:  [putdown]
+--no:  durationLessAfter(A,0.1) ?
    +--yes: increase(A) ?
    |       +--yes: durationMoreAfter(A,0.1) ?
    |       |       +--yes: [move]
    |       |       +--no:  [move]
    |       +--no:  durationLessBefore(A,0.1) ?
    |               +--yes: [move]
    |               +--no:  [putdown]
    +--no:  durationMoreBefore(A,0.1) ?
            +--yes: [pickup]
            +--no:  durationLessAfter(A,4) ?
                 +--yes: durationMoreAfter(A,3) ?
                 |   +--yes: [move]
                 |   +--no:  durationMoreAfter(A,2) ?
                 |       +--yes: [pickup]
                 |       +--no:  durationLessAfter(A,1)?
                 |           +--yes: [pickup]
                 |           +--no:  [pickup]
                 +--no:  [pickup]
```

Figure 5.6: Example of relational decision tree generated by Tilde.

5.5 Experimental Results

In the experiments, we have used the dataset described in 4.2.2.1. For estimating the accuracy 10-fold cross-validation was used. Table 5.1 shows the results obtained from the nine configurations tested in the experiments. As it can be seen, the accuracy in detecting the right action performed is quite high in the case of no noise, and it degrades as instances have more noise. It is still in acceptable accuracy levels even with the combined levels of noise of sensor failures and false negatives. It can also be seen that it degrades better with increasing percentage of false negatives than with increasing levels of sensor failures. The explanation for this behavior is that while each false negative affects always two examples, the sensor failure can affect more than two. When a sensor failure is simulated, the sensor's time counter is reset. This simulates that an object stops being detected just for a moment and then is being detected again. This behavior is common when holding an object and the object moves. In these cases, the RFID stops detecting the object for a moment. For that reason, the counter is reset. This is done automatically by the simulator. This counter holds the time that a tag was being detected, and it is used by the background knowledge to provide more information in order to create a better tree to classify.

Table 5.1: Average accuracy of the system.

false negatives / sensor failures	0%	5%	10%
0%	97.8	**98.1**	96.5
5%	84.0	86.6	84.4
10%	**74.0**	80.0	76.8

When false negatives increase from 0% to 5%, the performance improves. This increment produces more variety in the examples that help to deal better with the other kind of noise. As expected, the tree created in this case is bigger than the one created without false negatives. When the noise goes up to 10%, the performance degrades again. In this case, the noise affects the system performance, because it generated too many bad examples, making classification task harder. Table 5.2 shows the confusion matrix of the best case and Table 5.3 shows the confusion matrix of the worst case, both marked in bold in Table 5.1 that corresponds to the cases that obtained the highest and lowest accuracies. These good results show that the approach is a promising one.

Table 5.2: Test with no sensor failures and 5% of false negatives. It obtained the best results.

classified as $\longrightarrow$	move	pick-up	put-down
move	49.85	0	0
pick-up	0.62	24.45	0
put-down	0.62	0	24.45

Table 5.3: Test with 10% of sensor failures and no false negatives. It obtained the worst results.

classified as $\longrightarrow$	move	pick-up	put-down
move	43.45	3.79	6.12
pick-up	6.70	16.62	0
put-down	9.62	0	13.70

In both cases the classifier fails mostly when it predicts class *move* and it really was one of the other two actions. When the user *picks up* an object and then the sensor fails right after the object is picked up, the reported time by the sensor is so short that the learning system cannot differentiate between this instance, and another one in which a user is moving and passing close to the object (the detection time would be similar). In the case of *put-down*, the situation is similar, though the sensor failure occurs at the end of the action. The other kind of errors in classification come when the learning system classifies as *pick-up* or *put-down*, when it really was a *move* action. In these cases, the sensor failures occur while the user is moving around holding an object. If enough time has passed from the moment it picked up the object, the classifier will believe that the user has *put-down* the object in the moment that the sensor failed to detect the object. Also, if enough time has passed since the moment the sensor worked again and the moment in which the user has *put-down* the object, the classifier will label the instance in which the sensor works again as *pick-up*.

5.6 Discussion

This chapter presented an algorithm for activity recognition that uses relational learning in order to map changes in sensor readings to user's actions (low-level actions). The novelty of this approach relies on the fact that the algorithm does not try to recognize the activity (high-level activity) itself. Instead, it recognizes the low-level actions that the user performs to accomplish the activity (goal). The results show that our approach is a viable alternative. The classification of the low-level actions permits to recognize the high-level activities that users perform by grouping the actions.

The next step to reach our goals is to use the proposed system to recognize activities employing real data. To that end, we will use public datasets. Although the proposed approach in this chapter is promising, the learning system used (Tilde) has a limitation. It can not manage large amounts of data. After trying Tilde, we had to change the method in order to manage large datasets. For that reason, in the next chapter we change the approach and, instead of using a learning system based on a relational representation, we use a learning system based on a propositional representation to build an AR system capable of managing large amounts of data.

Chapter 6

Segmentation Algorithm Based on Events

Generally speaking, human activity recognition (AR) can be defined as the automatic recognition of an activity or a state of one or more persons based on observations coming from sensor readings. Usually, this is performed by following a fixed length sliding window approach for the features extraction where two parameters have to be fixed: the size of the window and the shift. In this chapter we describe a different approach using dynamic windows based on events. Our approach adjusts dynamically the window size and the shift at every step. Using this approach some models were generated to compare both approaches.

This segmentation method has been developed in order to find an approach capable of dividing the sequences of sensors' readings like planning operators would do it. Since the overall objective of this thesis is to generate a planning domain, in this work a different method for segmentation is explored considering that human actions behave like planning operators because, in the planning domain, the operators will represent human actions. As it was mentioned in Section 2.2, planning operators produce changes in the state of the world. Thus, with this approach the method proposed tries to find the changes produced by the actions of the user that we want to model as planning operators [Ortiz *et al.*, 2011].

In this chapter we take an alternative way to the one taken in the previous chapter. We set apart the relational model developed in the previous chapter since it can not manage large datasets and we start a new algorithm using real data and propositional models capable of learning on large amounts of data.

6.1 Introduction

Usually, AR problems are tackled as a machine learning problem where the observations collected by the sensors are the inputs, the performed activities are the classes, and the learning techniques generate classifiers. These classifiers will take as input new sensor readings and generate as output the predicted action the user just executed. Sensors produce data streams that can be seen as simple time series, a collection of observations made sequentially in time. So, the recognition system must process the inputs to extract the learning instances, their feature values and the classes. The features depend on the available sensors. Thus, in [Patterson *et al.*, 2005] RFID sensors are used and the features extracted are the RFID tags detected by the RFID reader. In [Kasteren *et al.*, 2008], two-state sensors are used and the features are the states of all sensors. Other types of sensors like accelerometers produce continuous data streams and the features must be extracted from those. For instance, the features extracted in [Bao and Intille, 2004] are the mean, energy, frequency-domain entropy, and correlations of the other features. In [Huynh and Schiele, 2005], they use similar features as well as the magnitude of the mean, variance, energy, spectral entropy, and the discrete Fast Fourier Transform (FFT) coefficients.

Independently of the sensors used, in the feature extraction step most AR systems use a sensory sequence segmentation based on a fixed-size sliding window [Bao and Intille, 2004; Tapia *et al.*, 2004; Stikic *et al.*, 2008a]. In those cases, many of the classification errors come from the selection of the sliding window length [Gu *et al.*, 2009]. For instance, an incorrect length may truncate an activity instance. In many cases, errors appear at the beginning or at the end of the activities, when the temporal window overlaps the end of one activity and the beginning of the next one. In other cases, the window length may be too short to provide the best information for the recognition process. In [Huynh and Schiele, 2005], the authors studied different features and window lengths. They showed that the best performance is achieved when different window lengths and features are chosen separately for each activity.

Besides, the static sliding window approach generates many identical consecutive temporal windows with exactly the same features and the same activity performed when the user executes the same activity during a long period of time. Those repetitive instances do not contribute to solve the problem better. Instead, they produce higher classification scores of the activities during which those instances are generated. But they do not help to recognize other activities, and the systems have to classify the identical instances over and over again.

For those reasons, we hypothesize that a different segmentation approach based on non-fixed length windows may achieve better results. Thus, we propose an approach based on events to generate dynamic sliding-windows to infer the activities. So, instead of defining a static fixed-length window, we define the events that will be used to define the boundaries of the dynamic windows employed to extract the features. Hence, when a specific event in the sensors readings is detected, we extract the features to classify what the user did between that event and the previous one. Those features are always the same, but the size of the windows changes based on when the events happen. So, the size of the window is dynamically established by the events. Thus, the windows are dynamic in time although the number of events in a window is always the same. In addition, our method does not create any temporal window if no events are detected.

The events we use are sensor dependent and domain-independent. In the case of RFID or reed switch sensors, an event could be any sensor state change. That is the case of the datasets used in this thesis. Using sensors producing continuous data like accelerometers, magnetometers, gyroscopes or GPS's, one or several thresholds could be set in order to detect the events.

The goal of this chapter is to learn the actions that users perform, using the dynamic window method based on state changes on public datasets, and to compare the results with other approaches. For our experiments, we used data from two different sources. The first dataset used was presented in [Kasteren *et al.*, 2008]. It uses a set of two state sensors deployed in a house. The second dataset is the one used in [Patterson *et al.*, 2005]. In this dataset, RFID readers and a set of RFID tags installed in the environment are used to detect the activities. Models are built using some state-of-the-art algorithms for classifying the activities, including also the models used by the authors of the datasets in order to compare their models and ours.

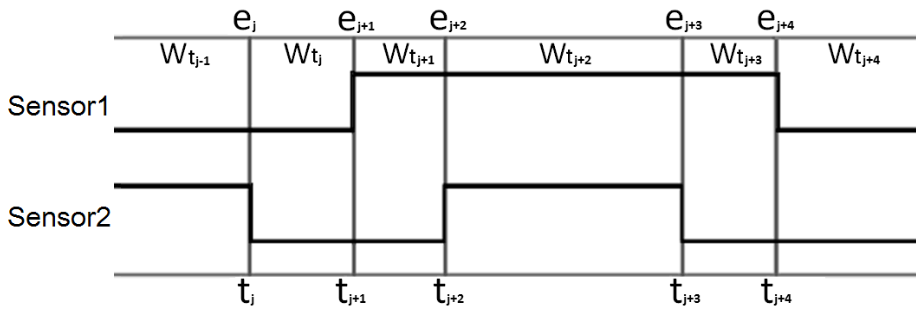

Figure 6.1: Temporal segmentation on a time series of two sensors by the dynamic window method.

6.2 Dynamic Windows Based on State Changes

The static sliding window approach uses fixed-length temporal windows that shift to create instances. Each window position produces a segment that is used to isolate data for later processing. A detailed description of the sliding window approach can be found in 2.1.2.1.

In contrast, our approach generates the learning instances, given by f_1, from a temporal window created by using as boundaries what we call *significant* events. So, it uses the last m *significant* events to generate the learning instances. Also, instead of sliding, it uses the next events to fix the boundaries of the next instance. Hence, the approach relies on the events detected by the sensors that the system uses instead of the window length. Figure 6.1 shows an example of our method using as significant events e_j all the changes in the values of the sensors.

Thus, our method does not set temporal values for l, the size of the window, and r, the shift. These values change over time. The function f_1 can be formulated as follows. Given N sequences of sensors readings as above, X^s, we generate one sequence of significant events $E =< e_0, e_1, .., e_j, .. >$ that are detected at time steps $T =< t_0, t_1, .., t_j, .. >$ where t_j is the timestamp of event e_j. The events are detected from all sensors readings, but they are merged into E. Then, the sequences of all sensors are segmented by the events $e_x \in E$, from all sensors. Those events will divide the sequences in Z temporal windows $W^s_{t_j} =< x^s_{t_j}, .., x^s_{t_{j+m-1}} >$ where $W^s_{t_j}$ will contain all the readings of sensor s from t_j to t_{j+m-1}, being m the number of significant events used to create the windows. The next window will be defined as $W^s_{t_{j+1}} =< x^s_{t_{j+1}}, .., x^s_{t_{j+1+m-1}} >$ so the shift at every step will be set dynamically as $r_{j+1} = t_{j+1} - t_j$. Then, $W_{t_j} =< W^1_{t_j}, .., W^N_{t_j} >$ will be the segments of the N sensors at time t_j. Figure 6.1 shows these segments W_{t_j} where $m = 2$. Events will divide the sequences of all sensors, even the sequences of the sensors that did not detect such event.

Once the temporal windows are delimited, the features $\vec{F_{t_j}}$ are extracted and they are labeled with an activity $a_j \in A$. The activity that will label the window W_{t_j} will be the activity that the user performed between the last two events t_{j+m-2} and t_{j+m-1}. So, we are assuming that all changes in activities are detected by at least one sensor. Thus, the user can not be performing two activities in the same window. Finally, the function f_2 is executed.

One of the main differences between the two methods is that our approach generates a new window just when a new event is detected, whereas the other approach continues creating new windows even when the sensor readings do not change; temporal windows created when there are no changes in the sensors are identical. It can be seen in Figure 2.1 where the window W_{i+3r} does not contain any event, so the vector $\vec{F_{i+3r}}$ will be identical to the vector $\vec{F_{i+2r}}$ generated by the previous window W_{i+2r}.

6.3 Evaluation

In order to test the segmentation method, we generated models based on the sensors used to record the two selected datasets [Kasteren *et al.*, 2008; Patterson *et al.*, 2005]. These datasets were selected since they are well known and used by the community. Each dataset employed different sensors and the method is sensor dependent. For that reason, we chose different *significant* events to generate the classification instances. We first generated models following the dynamic window approach. Then, we compared our models with the fixed-length sliding-window approach employed by the authors. Next, we describe the models generated for our experiments in detail.

6.3.1 Kasteren Models

Three configurations were tested with this dataset. In the first one, we reproduced the model that achieved the best results in [Kasteren *et al.*, 2008]. They employed temporal probabilistic models and divided the data in slices of constant length, 60 seconds, without overlapping. So, the parameters used were $l = 60$ seconds and $r = 60$ seconds. A vector of features was generated for each slice. The vector contained one entry for each sensor, where the values of the sensors could be `0` or `1`. They tested four configurations. We will focus on the one that got better results. In that configuration they used two representations in parallel that they called *change point* and *last*. In the *change point* representation the sensor gives a 1 to time slices where the sensor reading changed. In the *last* representation the last sensor that changed its state continues to give 1 and changes to 0 when a different sensor changes state. In our experiments we recreated this in the first configuration but we used a Dynamic Bayesian network (DBN) equivalent to the Hidden Markov Model (HMM) used by them. To do so, we added the id of the last activity performed to the vector of features.

In order to test our approach, we built some models using the same representation over the same dataset, but generating the instances using the dynamic window approach and using a different representation. To use this approach, we considered as a significant event any change in the sensor readings. That is, when a sensor changes its state from `1` to `0` or vice versa. Since we used changes in the sensors readings instead of slices of constant length, we generated our instances from the last `10` changes, $m = 10$. A way to select the length of the temporal window is using the average time spent performing an activity in the data, as in [Patterson *et al.*, 2005]. Instead of using the time, we used the events. So, we divided the state changes of the dataset, `2638`, by the number of activities `245` and we obtain $m = 10$.

There were 14 sensors, so the features were a vector $\vec{F_{t_j}} =< F_j^1, ..., F_j^{14} >$ for the event e_j where $F_j^n \in \{0, 1\}$. The F_j^n was set to 1 if the sensor n changed its state at least once between t_j and t_{j+9} $(m = 10)$. Also, we kept to 1 the last sensor that changed its state. This way we reproduced the representations *change point* and *last* used by the author. Additionally, we added the id of the last activity performed to the vector of features to learn activity transitions like the HMM does. This configuration was called *DSW-1-K* for

Dynamic Sliding Window using Kasterens dataset. For the last configuration, *DSW-2-K*, we created a new model using a different representation and features. Instead of using the state of all sensors during the temporal window, whether the sensor changes its state or not, in this model we construct the vector using the identifier of the sensors that produced the last 10 events. So, the features of the segment that starts in e_i are $\vec{F_{tj}} =< s_{t_j}, ..., s_{t_{j+9}} >$ where $s_{t_n} \in \{1, ..., 14\}$ since there were 14 different sensors. s_{t+9} is set to the identifier of the sensor that produced the event e_{j+9} and s_{t_j} is the identifier of the sensor that produced the event e_j. If a sensor is not responsible for any event in the last 10, it is not included. Also, if a sensor produced k events, the identifier of that sensor is included k times. So, for example, if in the current temporal window W_{t_j} the sensors that changed its state were $1, 2, 3, 4, 1, 2, 3, 4, 5, 6$ then the features used would be $\vec{F_{t_j}} =< 1, 2, 3, 4, 1, 2, 3, 4, 5, 6 >$.

Then, we tried to reduce the number of features extracted from the sensors performing a feature selection method by computing the Information Gain Ratio [Hall and Smith, 1998] for each of the features and then ranking them from highest to lowest. Afterwards, we created and tested a model using all the features. The worst feature according to the ranking was eliminated and a new model was generated and tested with the remaining features. All the features were progressively eliminated until none is left. All generated models are compared and the features employed in the best model were kept. Using this feature selection method, we reduced the size of the vector from 10 to the last two sensor state changes $\vec{F_{tj}} =< s_{t_{j+8}}, s_{t_{j+9}} >$ being $s_{t_{j+8}}$ and $s_{t_{j+9}}$ the id of the sensors that detected the last 2 events. Finally, we added the id of the last activity performed like in the other models.

Since we are interested in comparing probabilistic models with others models more easily readable by humans, *DSW-1-K* and *DSW-2-K* were generated using *PART* [Frank and Witten, 1998], an algorithm that generates rule sets, and *J48* [Quinlan, 1993], a decision-tree learning algorithm, instead of a DBN.

6.3.2 Patterson Models

We also tested three configurations using this dataset. In the first configuration we used the fixed-length sliding-window approach used by the authors of this dataset. We also used their features and representation. They divided the data in slices of constant length, where the mean length of each uninterrupted portion of the interleaved tasks was 74 seconds. At each second they generated a vector of features with the data of the last 74 seconds. So, the parameters used were $l = 74$ seconds and $r = 1$ seconds. The vector contained 74 entries, one for each second, where the values of each entry could be *object-X-touched* when an object was detected or *no-object-touched* when no objects were detected. They used temporal probabilistic models too. They tested four different models. We have reproduced one of the most accurate models they generated; a HMM equivalent to the one used by Kasteren in the previous dataset. So, in order to replicate the results we created a DBN equivalent to the HMM used by the authors as with the Kasteren dataset.

In *DSW-1-P*, we used our approach with the same representation. We used a vector with the last 74 significant events to recognize the activity that was performed between the last two events. In this case, we considered as significant event when the RFIDs change the detected object or no objects are detected. So, the features used were the id of the new object detected or *no-object*. The dataset just provides information about the objects detected and the timestamps. Therefore, to detect the *no-object* state we assumed that when two readings are found in the dataset and the time interval between them is one second or more there is at least a state with no objects detected.

We also generated a new model using a simpler representation in *DSW-2-P*. We used a vector of features composed of the last 74 events like in the previous setup. Then, we applied the same feature selection method that we used in the previous dataset to find the optimal combination of features. That way we obtained a very accurate model using just the last two events instead of 74.

Like in the other dataset, we generated the models that use our approach, *DSW-1-P* and *DSW-2-P*, employing the *J48* and *PART* algorithms. We added the activity performed previously to the vector of features used in this experiment to learn activity transitions like we did in the previous dataset.

6.4 Experimental Results

In summary, we have tested six configurations, using the models described in the previous section. We used two metrics to evaluate the models using 10-fold cross-validation for estimating the error: precision and recall averaged over all activities. We have used these two metrics instead of accuracy, used by the authors of the datasets, because the datasets are unbalanced. So, accuracy is not a good metric as described in [Chawla, 2010]. Hence, we have used precision and recall as recommended in [Kasteren *et al.*, 2010c; Chawla, 2010]. What we call precision was used as a metric by Kasteren but he called it *Class Accuracy*.

The learning algorithms we used were *DBN* to learn the models that replicate the models used by the authors of both datasets and *J48* and *PART* to learn our models.

Table 6.1 shows the average precision and recall obtained by each setup, and the number of instances generated by each model. The best results have been marked in bold. In addition, we have included a column with the percentage of times that the temporal window generated by the segmentation method contains at least one event able to change the state of the sensors and produce an instance different from the one produced in the previous window. That is, any change in the sensors readings that generates one instance different from the previous one. This counter measures the diversity of the temporal windows from which the instances are created since the values of the sensors in the temporal windows without state changes will be the same as in the previous window.

Table 6.1: Precision, recall, number of instances and diversity for all setups.

	Precision	Recall	N. Instances	Diversity
Kasteren DBN	80.55	80.08	40003	3.16%
DSW-1-K J48	92.16	91.15	2638	68.35%
DSW-1-K PART	92.28	90.85	2638	68.35%
DSW-2-K J48	**93.05**	91.34	2638	68.35%
DSW-2-K PART	92.61	**91.38**	2638	68.35%
Patterson DBN	78.90	86.57	16280	27.16%
DSW-1-P J48	94.80	94.48	5408	100%
DSW-1-P PART	**97.72**	**96.76**	5408	100%
DSW-2-P J48	94.80	94.49	5408	100%
DSW-2-P PART	97.69	95.32	5408	100%

As we can see from the results of both datasets, the precision and recall of the static sliding-window approach, *Kasteren DBN* and *Patterson DBN*, are much lower than the results using a dynamic window, *DSW-1-K*, *DSW-2-K*, *DSW-1-P* and *DSW-2-P*. The table shows that the different models generated using the dynamic window approach obtain similar results. The feature selection improved slightly the results in the first dataset, though not in the second dataset. The precision of *Kasteren DBN* is 80.55%, quite similar to the result reported by Kasteren in [Kasteren *et al.*, 2008] which is 79.4%. The DBN we used needs some parameters to be defined, so the difference probably is due to dissimilarities in those parameters settings. We can not compare our results with those reported by Patterson in [Patterson *et al.*, 2005] since they do not report on precision and recall.

The table shows as well that the number of instances created by each method is very different. The static sliding-window approach creates many more instances than our approach in both datasets. Our method creates an instance just when a specific event is detected, while the static sliding-window approach creates instances for every time slide even when the user is not at home, or is sleeping and no events are detected. That is the case of the first dataset. The behavior in the second dataset is similar; many instances are created when the system does not detect any event, since one instance is created every second. So, most of the instances are generated without changes in the sensor readings. This fact is shown in the last column of the table where our models generated higher percentages of different temporal windows. Notice that the diversity of the dynamic window in the first dataset is not 100% like in the second dataset because some of the significant events produced the same feature. When a sensor changed its state from 1 to 0 or vice versa the value of the feature extracted was 1 in both cases because of the *change point* representation. Thus, some consecutive generated instances were identical. In any case, using near 5% of the instances generated by the sliding window approach, we obtain better precision and recall.

A deeper analysis of the models shows that the sliding window fails more often when the activity changes. Comparing the *Kasteren DBN* and *DSW-2-K PART* we see that the first one fails in a 96.37% of the instances when the activity changes, whereas the second one fails in 48.12%. The results are similar in the second dataset, so *Patterson DBN* fails in 86.58% whereas the *DSW-2-P PART* fails in 42.13%. Thus, the activities with fewer instances reach a lower precision and recall, whereas the activities with many instances like *leave house* and *sleep* in the first dataset and *Making soft-boiled eggs* in the second obtain better results.

The experiments show that the different classification algorithms obtained similar results. Although experiments with the Pattersons dataset show that PART obtained better results than J48, the differences are small in both datasets.

6.5 Discussion

In this chapter we have presented a different approach to create the learning instances for AR. The novelty of this approach relies on the fact that our system uses the changes in the information captured by the sensors to create the instances for classification instead of the temporal sliding-window approach. We have compared our approach in public datasets used in the past by other researchers and we have shown very good performance. The results show that the approach obtains higher scores in precision and recall in the datasets used to test it.

The main advantage of the dynamic window approach is that it provides accurate models using much lower number of learning instances and features. This makes this approach suitable to be used online and in situations where computation times are important, since fewer instances would be evaluated and the instances will be processed faster.

One limitation is that instances depend on the sensors accuracy. So, whenever the sensors do not capture a significant change in the environment, the system does not detect the state change and it does not create the corresponding instance. Anyway, this case is equivalent to the case when the temporal window is too long and contains two activities instead of just one. In [Logan *et al.*, 2007] the authors have shown some of the limitations of RFID in extensive use. However, other sensor modalities like accelerometers or motes can be used to recognize the activities.

We have described how models can be used for AR employing this method to extract the information from the sensor readings. The generated models are able to predict transition probabilities better by recording the last objects observed in each activity. So, good results can be obtained by using just the last two objects detected by the RFID or the last two reed switch sensor that changed the value. While there are still technical challenges to overcome, this work shows that AR using dynamic windows to generate the instances and just some of the changes in the states of the sensors as features can be a good choice.

The method is inspired in the way planning operators work. When a planning operator is executed, the world changes. For that reason, the changes in the sensor readings are used for segmenting the sensor data. Using *significant events*, we tried to match the events produced by the sensors and the execution of planning operators. That way, the segmentation would prepare the data from the beginning to be used to generate planning operators. Unfortunately, the activities performed by humans usually generate many changes in the sensor readings during a short period of time and not all the events produced during that period of time belong to a single activity.

Chapter 7

EBAR: Event Based Activity Recognition

In this chapter a new algorithm for the recognition of human activities called EBAR (Events Based Activity Recognition) is introduced and evaluated. The starting point of the process is the event based approach to extract features from time series of sensors presented in the chapter 6. This segmentation method, in contrast to the commonly used sliding-window method, allows EBAR to use new features to improve the results of the recognition process. These features provide information about the boundaries of the activities and are used along with common features to obtain a better recognition of the activities particularly in those instances placed in the boundaries of the activities. Experiments with public datasets show that our algorithm is able to: accurately recognize the activities using less instances than other approaches; recognize the boundaries of the activities better than other approaches; and also adapt itself to changes in the user behavior, and still obtain good results.

7.1 Introduction

We are interested on building an AR algorithm able to particularly recognize the start and end of activities, because its output will be used to automatically learn user behavior models. In this chapter a complete algorithm for activity recognition is presented. We build on our prior work presented in Chapter 6 and extend it by studying more deeply the segmentation methods and including new features to better classify activities.

Usually, the events that enclose the activities occur also during the activities. For that reason, we propose to use a pre-classification step to determine whether a significant event bounds or not an activity. Once a boolean class is assigned to the significant event, the result is added as a new attribute to the rest of features extracted and generated in the previous steps to recognize the user activities in the final step. In this work, a new algorithm is also proposed to integrate the features described above into a framework to learn the actions that users perform. The resulting algorithm is called EBAR. The results are compared with other approaches using public datasets. We build models that use state of the art algorithms for classifying the activities. In addition, we also present results on how the shift and the length of the temporal windows affect the recognition task.

7.2 The Activity Recognition Algorithm: EBAR

Our approach covers the entire recognition process from receiving sensor readings to the final matching between the time series temporal window and the activities performed in such intervals. The following sections describe the steps EBAR takes that are different from previous works. Figure 7.1 shows the system's architecture where the input are the sensors' readings and it outputs the model to classify activities.

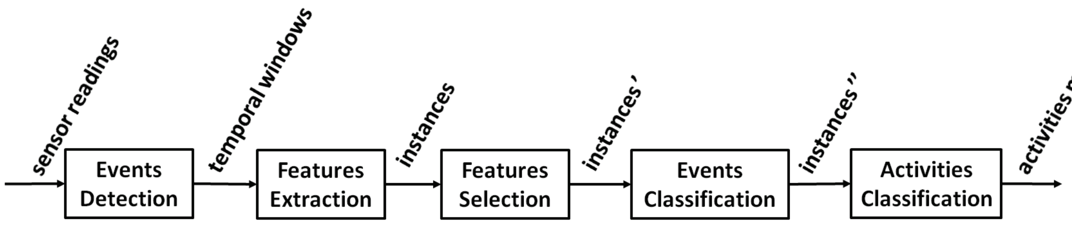

Figure 7.1: EBAR's architecture.

7.2.1 Algorithm Description

EBAR employs temporal windows created by using as boundaries significant events to generate the learning instances. It uses the dynamic windows approach (Section 6.2). Figure 7.2 shows an example representing the dynamic window approach using as significant events all the events produced by sensors 1 and 2, and sliding-windows without overlapping. In this case, we define as significant event every time a sensor changes its state from the lower level to the upper one or vice versa and $m = 1$. In the example our method generates six temporal windows that will be used to extract the classification features for f_2. It adapts each window length to the events produced by the sensors. Instead, the static sliding-window method generates eight windows, all of them of the same size, even when nothing happens (windows 2, 5 and 6).

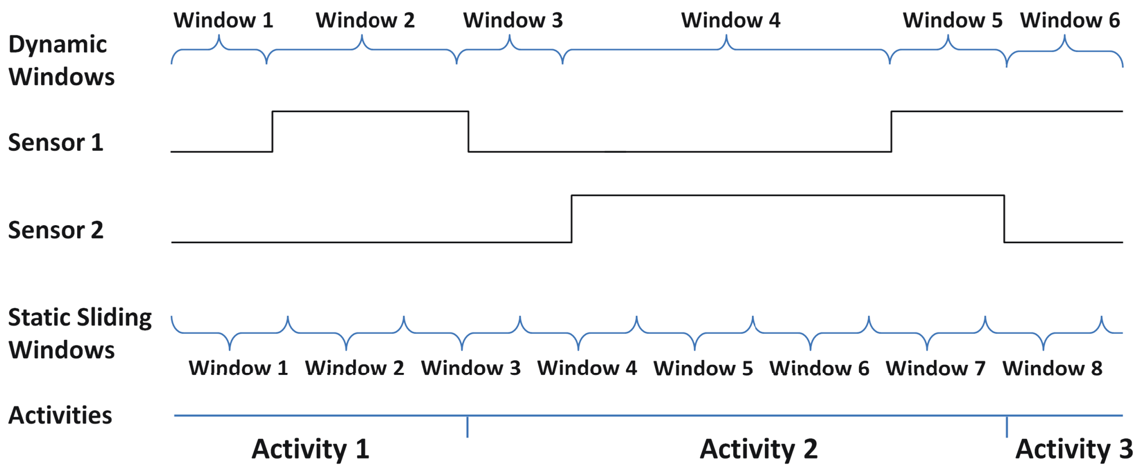

Figure 7.2: Dynamic windows versus static windows.

The significant events must split the whole dataset in n or more temporal windows, where n is the number of performed activities. For example, if a subject executes four different activities, a significant event approach for this dataset should split the data into four or more temporal windows. For each of the four activities, our approach would find one of these events at the beginning and another one at the end. It might also find one or more in between. Thus, the perfect significant events for this dataset will generate four temporal windows to classify. The worst case would be the one where an activity does not generate any event. In this case, EBAR would create, using the next significant event, a temporal window which would contain two activities. Thus, less than 4 temporal windows may be created. Then, it would behave like the fixed-length sliding-window method when a temporal window contains two or more activities.

For instance, using a sensor network composed of RFID sensors, the events the system would use could be defined as any change in any sensor reading. So, every time the sensors report a new reading, it is compared with the previous reading reported by the same sensors. If the two readings are different, a new event is found and a new temporal window is created, associated to the event. As another example, using the same sensor network, the significant events could be defined as specific readings, such as when the sensors detect a given object. Every time this object is detected is considered as an event and a new temporal window is created. Any event that occurs at the beginning and at the end of every activity is suitable to be used. Thus, the way to define the events is to select one of these. In our approach the significant events are not defined arbitrarily. Instead, training data is used to find out those events. Consequently, the training data is used for two purposes: to train a classifier to recognize activities; and to learn to detect significant events. For example, using the Kasteren dataset described in Section 4.2, it can be found that every time the user changes the activity a sensor fires up. That is, it changes its state from 0 to 1. In this case, those changes can be used as significant events or alternatively all the changes can be used, since both match the boundaries of the activities. In the case of analog sensors, like in the Huynh dataset, a threshold is calculated empirically. We first compute the difference in the sensors readings when the user changes the activity. Then, the minimum value in the difference is used as a threshold. A value higher than the minimum one would skip some of the significant events.

Significant events divide the sequences of all sensors, even the sequences of the sensors that did not create such event. For instance, two of the three datasets used for evaluating the algorithm, the Kasteren [Kasteren *et al.*, 2008] and Huynh [Huynh *et al.*, 2008] datasets, employ more than one sensor. In the case of the Huynh dataset, there are two accelerometers and the sequence of an accelerometer would be divided by the significant events produced by the other accelerometer. Kasteren uses binary sensors, and, whenever a sensor generates a significant event, the states of all sensors are used to generate the features that compose the instances. This is not new since the fixed-length sliding-window method also includes the information of all sensors in every temporal window. That way, we provide as much information as possible to the algorithm. Later, the attribute selection and ML algorithms will select the best features to classify the activities.

Once the temporal windows are delimited, the features are extracted and labeled with an activity. The activity that will label the temporal window will be the activity that the user performed between the last two significant events of that temporal window. Next, EBAR reduces the number of features and includes a new one in order to obtain the best recognition rate. Finally, a machine learning algorithm is used to learn a classifier. The whole process is described in Algorithm 2.

Our AR algorithm is composed of two main stages. The first one, f_1, captures, cleans,

Algorithm 2 EBAR Algorithm.

```
Input: m, $mlAlgorithm_1$, $mlAlgorithm_2$
Output: mlFinalModel (Activities classifier)

  numberEvents ← 0
  buffer ← empty
  instances ← empty
  sensorR ← sensorAction ()
  while sensorR $\neq$ null do
    sensorRCleaned ← preprocess (sensorR)
    buffer ← buffer + sensorRCleaned
    if (isSignificantEvent (buffer)) then
      numberEvents ← numberEvents + 1
      if (numberEvents = $m$) then
        numberEvents ← 0
        instance ← featureExtraction (buffer)
        instances ← instances + instance
        buffer ← sensorRCleaned
      end if
    end if
    sensorR ← sensorAction ()
  end while
  instances' ← featureSelection (instances,$mlAlgoritm_2$)
  mlModel ← learn (instances', $mlAlgoritm_1$)
  instances" ← eventClassification (mlModel,instances')
  //End of $f_1$ and Beginning of $f_2$
  mlFinalModel ← learn (instances", $mlAlgoritm_2$)
  return mlFinalModel
```

segments and processes sensors data to create the instances to be classified. The second stage, f_2, uses these instances to predict the activity the user is performing. The first stage begins with capturing raw data from sensors as well as the action (class) *sensorAction()*. Incomplete and erroneous readings are removed from the raw data in *preprocess (sensorR)* to generate clean data, *sensorRCleaned*. The results of this step are sensor time series ready to be processed. In the case of public dataset, like the datasets we are going to use for testing, the data has already been cleaned so this step is skipped. Next, these sensor time series are segmented using a dynamic sliding-window method, presented in Section 6.2. The method uses *isSignificantEvent(buffer)* to compare the current reading, *sensorRCleaned*, which is the last element in *buffer*, with the previous readings, rest of *buffer*, in order to detect significant events. Then, if the number of significant events equals m, which is the number of significant events making up a segmentation window, a temporal window is used to extract features, *featureExtraction(buffer)*, where *buffer* contains the data of the temporal window. As a result one instance is created. The value of m is determined empirically as we will describe in Section 7.3. Once the segmentation process has generated all the instances from the sensor data, the set of features is reduced by an attribute selection algorithm in the next step, *featureSelection(instances)*. A wrapper method is employed [Kohavi and

John, 1997] along with the machine learning algorithm that is going to be used to learn the activities classifier. This way we get a feature subset tailored to the data.

Next a double classification process is performed. The first classification, *learn(instances', mlAlgoritm*$_1$*)*, generates a model, *mlModel*, to determine whether a significant event bounds an activity or not. Once that model is obtained, it is used to assign a boolean class to each instance and the result is added as a new attribute to the rest of attributes of the instance. Then, the activity recognition process is executed, *learn(instances", mlAlgoritm*$_2$*)*, to generate the AR model. At learning time, the sensor reading captured by *sensorAction()* has two classes associated, whether the readings correspond to a boundary event and the activity performed. The difference between the two ML processes is that the first one does not use the activity class, so it uses one less attribute, and the last one uses one more attribute, the boolean class that is the result of the previous process. Both steps can be executed using any supervised machine learning algorithm.

7.2.2 Events Classification

Nothing prevents significant events from appearing also inside activities. Once a significant event is detected, a classification step is performed to determine whether the event really bounds an activity or not. The result of this classification is a series of *boundary events*, significant events that really enclose an activity. These boundary events could be used to segment the sensor data, but experimental results using this segmentation are not good. As it will be shown in Section 7.4.2 the accuracy obtained in the event classification is not good enough to rely on it to generate the windows. Instead, we use any significant event to generate the windows, and add a new feature, whether we believe the last significant event is a boundary event or not, to help in classification.

Figure 7.3 shows an example where the significant events that enclose the activities occur when a sensor changes its state from the lower level to the upper one. In this example, three significant events would be detected and used for segmentation. Notice that there are two events where the sensor changes its state from the upper level to the lower one. These two events are not considered significant since none of them enclose an activity, thus we selected only the opposite state change. The three significant events generate four windows using the dynamic window approach. It can be seen that *Event 1* does not happen between activities unlike *Event 2* and *Event 3* do. Thus, in order to classify complete activities, EBAR tries to classify the events to separate the good ones, *Event 2* and *Event 3*, from the bad ones, those that occur during the activities, like *Event 1*. Once the significant event is classified, the result is included in the instance created by such event as a new attribute.

7.3 Evaluation Setup

In order to test EBAR, some models have been generated using the three public datasets previously presented [Kasteren *et al.*, 2008], [Patterson *et al.*, 2005] and [Huynh *et al.*, 2008]. These datasets include different types of sensors.

In the first dataset the same representation and features presented by the authors were used, but the instances were generated following our segmentation method. The results have been compared with the fixed-length sliding-window approach used by the authors. In addition, the representation and the attributes were changed to test other options in order to obtain even better results. In the second dataset the dynamic sliding-window approach was also used with two different representations, one of them being the one used by the

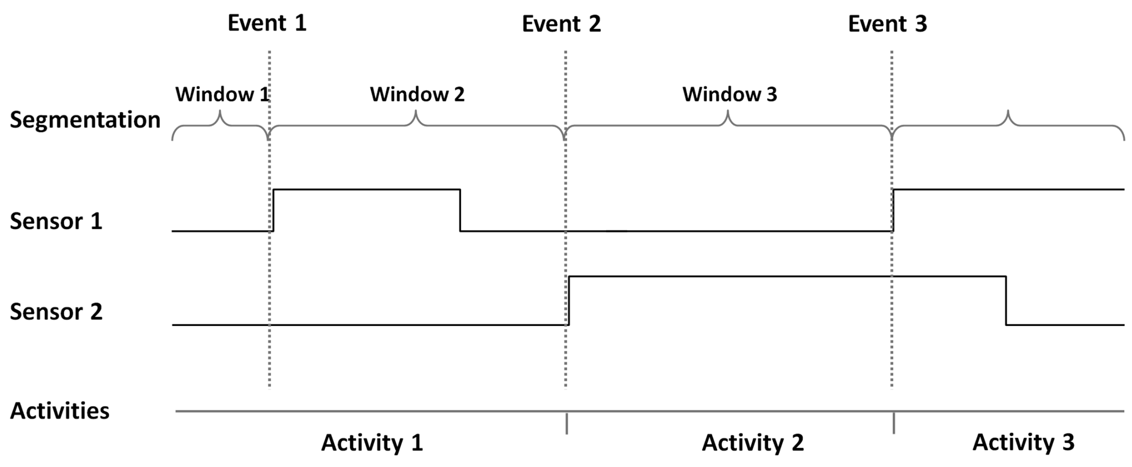

Figure 7.3: Example of segmentation.

author of the dataset. For the third dataset, models were generated using EBAR and results were compared to a fixed-length sliding-window approach using the same features.

In addition to the algorithm used by the authors, other classifiers were used: *PART* [Frank and Witten, 1998], *J48* [Quinlan, 1993], *k-nearest neighbor* (kNN) [Aha *et al.*, 1991], *Random Forest* (RF) [Breiman, 2001], *Support Vector Machines* (SVM) [Platt, 1999], *Naive Bayes* (NB) [John and Langley, 1995], *Hidden Naive Bayes* (HNB) [Zhang *et al.*, 2005], *Additive Logistic Regression* (ALR) [Friedman *et al.*, 2000], and *Decision Tables* (DT) [Kohavi, 1995]. These algorithms have been chosen in order to test the performance of EBAR employing different types of ML algorithms although any other algorithm could be used. Their implementation in the Weka toolkit [Witten *et al.*, 1999] has been used with their default parameters.

7.3.1 Kasteren Dataset and Models

A detailed description of this dataset can be found in Section 4.2. Three configurations were tested using this dataset. For the first configuration the fixed-length sliding-windows is used and for the other two configurations, models were generated using the dynamic-sliding window. The first configuration is a replication of the original one and it is the same we described in Section 6.3.1. The second and third configurations use the same attributes and representation described in Section 6.3.1 in the configuration we called *DSW-2-K*. The second configuration *Kas-EBAR* shows the final results obtained by EBAR. The third one *Kas-EBAR W-Ev* shows the results of using the algorithm without including the step of classifying events. Thus, the results are shown with and without the events classification in order to test the usefulness of such step of the algorithm.

7.3.2 Patterson Dataset and Models

A detailed description of this dataset can be found in Section 4.2. Three configurations were tested using this dataset too. In the first one, *Patterson DBN*, the fixed-length sliding-window approach used by the authors of this dataset was employed. Also, the same features and representation of the authors were used and it is the same we described in Section 6.3.2.

In the next two configurations, *Pat-EBAR* and *Pat-EBAR W-Ev*, EBAR was used with the same representation and attributes described in Section 6.3.2 in the configuration we called *DSW-1-P*. As in the other dataset, in *Pat-EBAR W-Ev* the results obtained without including the classified event as an attribute are shown. In *Pat-EBAR* all phases of EBAR were used. The id of the last activity performed was added to the vector of all these setups to create the DBN of the author of the dataset and to learn the activity transitions. For all setups, the models were generated using the ML algorithms described in 7.3 and all of them were used in all the setups with the same features for each setup.

7.3.3 Huynh Dataset and Models

A detailed description of this dataset can be found in Section 4.2. In Chapter 6 we tested our segmentation method. In this chapter, we are testing a complete AR algorithm to be used to recognize the actions that we want to model as a planning domain. For that reason, in this chapter we include a new dataset to better test EBAR using continuous data in addition to the other two datasets.

Two experiments were executed. The first one replicates what the authors of the dataset did. It contains configurations using the static sliding-window approach for segmentation using different sizes for the windows. From the acceleration signal they computed the mean and variance of the three axes of the accelerometers over sliding-windows of sizes between 0.4 and 4 seconds. Also, they added the timestamp of the last reading of the temporal window as an attribute. A DBN equivalent to their HMM was used in this experiment in order to replicate the results of the authors of the dataset.

EBAR was used in the second experiment. We used as features the same ones as in the original experiment; the mean and variance of the three axes of the accelerometers and the timestamp of the last reading of the temporal window. This dataset contains continuous data, so defining the significant events was more difficult. We defined a threshold and a reading was compared with the next one, and each time the difference between the values of the data of the two readings was higher than the threshold, a temporal window was generated. Several configurations were tested using different thresholds over the values of the data provided by the dataset. This experiment used between 2 and 20 sensor readings, comparing the difference between the maximum and the minimum value of the readings. The best results were achieved with three sensor readings ($m = 3$). We also tried different thresholds. The results can be seen in Table 7.11. The ML algorithms described in 7.3 were used to generate models in this experiment.

7.4 Experimental Results

This section presents the experimental results of applying the two phases of the algorithm. First, we present the results obtained classifying the instances in the boundaries. Next, we show the results of the event classification. Then, we present the final results obtained after applying the complete algorithm.

7.4.1 Results Classifying the Instances in the Boundaries

As it was said before, the main objective of this chapter is to generate models able to more accurately recognize the instances generated in the boundaries of the activities. The instances where the activity changes are the most difficult to classify. Also, the correct classification of these instances is very important in order to detect all the effects the activity produces to correctly create planning domains. In these cases, an analysis of the models shows that the sliding-window approach fails more often when the activity changes. While *Kasteren DBN* obtains a precision of 11.73% when the activity changes, *Kas-EBAR* achieves precisions from 16.04% using NB to 87.03% using kNN (k=1 which is the default value in Weka). The results are similar in the second dataset, where *Patterson DBN* obtains a precision of 13.42% and *Pat-EBAR* obtains results ranging from 26.85% using NB to the 89.35% reached by HNB. In all cases, and independently of the ML algorithm used, EBAR obtains better results than the fixed-length sliding window approach.

For the third dataset, *Huynh DBN* achieves a precision of 32.68% while EBAR obtains results ranging from 0.76% using J48 to the 99.34% reached by kNN. *Huynh DBN* obtained slightly better results than the worst model generated by EBAR. However, EBAR achieved a precision higher than 99% with one of the models. In this case, the segmentation of both configurations is equivalent, so the same temporal windows were generated. Both segmentations are equivalent because the threshold used by EBAR is so small that all readings generate significant events and the size of the windows used by Huynh is so small that all readings are used as different temporal windows as well. In this case, the attribute selection and the event classification do not help EBAR to classify the boundaries better. For that reason, the results obtained by *Huynh DBN* and EBAR DBN are the same.

Table 7.1: Boundaries classification results obtained using the author's models.

Models	Precision
Kasteren DBN	11.73
Patterson DBN	13.42
Huynh DBN	32.68

Table 7.2: Boundaries classification results obtained using EBAR.

	DBN	PART	J48	HNB	NB	LB	DT	RF	kNN	SVM
Kas-EBAR	16.38	58.70	61.77	65.87	16.04	41.63	59.38	85.66	**87.03**	61.09
Pat-EBAR	45.37	57.87	60.18	**89.35**	26.85	62.03	64.35	87.96	88.42	84.72
Huy-EBAR	32.68	5.97	0.76	-	23.01	-	69.16	95.00	**99.34**	-

Table 7.1 shows the results obtained by the author's DBNs and Table 7.2 shows the results obtained by EBAR. Both tables show the precision obtained. The best results are marked in bold. In the third dataset, three of the algorithms could not be executed due to the size of the dataset. It can be seen that the best results obtained in the three datasets are achieved by different ML algorithms. Results vary depending on the ML algorithm used. The differences in the ML algorithms are very big. Thus, choosing the right ML algorithm is very important in order to obtain the best results although algorithms like kNN or RF achieved very good results in all datasets.

7.4.2 Results of the Events Classification Phase

As we have said before in Section 7.2.2, we are especially interested on detecting the events that enclose activities in order to determine when an activity begins and ends. Tables 7.3 (Kasteren dataset), 7.4 (Patterson dataset) and 7.5 (Huynh dataset) show the confusion matrix obtained in the events classification phase of EBAR using the HNB algorithm with the first two datasets and kNN with the third one, which obtained the best results. The first row shows the instances of the significant events that bound the activities and the second row shows the rest of significant events. The first column indicates the instances that are classified as boundary events and the second column the instances classified as other type of events.

Table 7.3: Confusion matrix using the Kasteren dataset.

Boundary Events	Other Events	⟵ classified as
154	139	Boundary Events
45	2299	Other Events

Table 7.3 shows that good results were obtained, with an accuracy of over 93%. However, EBAR misclassified 47.44% of the events that we are interested in; those that occur at the beginning or end of activities. We have called *Boundary Events* to these events in the tables and the rest of the events are represented as *Other Events*. Table 7.4 presents better results than Table 7.3, but EBAR still misclassified 25% of the boundary events. Table 7.5 shows the confusion matrix of the third dataset that obtains the best results, with an accuracy of over 99.8%. However, it classified correctly less than 1% of the significant events in the boundaries. This is due to the high number of *Other Events* contained in that configuration.

Table 7.4: Confusion matrix using the Patterson dataset.

Boundary Events	Other Events	⟵ classified as
161	55	Boundary Events
2	5190	Other Events

Table 7.5: Confusion matrix using the Huynh dataset.

Boundary Events	Other Events	⟵ classified as
2	919	Boundary Events
26	772870	Other Events

As we said, we are interested on these events, because we want to find the boundaries of the activities in order to recognize entire activities, instead of pieces of activities as most of the AR literature does. Thus, the over-segmentation that occurs when the temporal windows of data are extracted from the sensors readings should be reduced. Also, detecting the beginning and end of one activity allows computing the duration of the activity, which in turn may help in the recognition process. In [Kasteren *et al.*, 2010b; Mckeever *et al.*, 2010] the authors show that modeling the duration of the activities improves the results obtained in the classification. The classification of the boundaries of the activities of our algorithm is far from being perfect. Hence, we included it as an attribute for the final classification step of recognizing actions. Next Section 7.4.3 shows that this attribute may help classifying the activities.

7.4.3 Classification Results Using Discrete Data

The first two datasets were generated using sensors that provide discrete data, while the third one included sensors that generate continuous data. In this section the results obtained using only the first two datasets are shown. Summarizing, six configurations were tested with the first two datasets: *Kasteren DBN*, *Kas-EBAR*, *Kas-EBAR W-Ev*, *Patterson DBN*, *Pat-EBAR* and *Pat-EBAR W-Ev*. For the experimental evaluation, the models described in Section 7.3 were generated. Two metrics have been used to evaluate the models using 10-fold cross-validation for estimating the error: precision and recall.

The novelty of EBAR is the addition of the classified significant event to the process along with the use of a dynamic sliding-window segmentation method. So, the performance of the models generated just before the addition of the classified significant event was tested to see if it is useful. Table 7.6 shows, for each algorithm, the average precision and recall obtained by the models in each configuration. It also shows the number of instances generated by each model. In addition, a column measuring the percentage of instances created that are different from the precedent one was included (diversity).

As it can be seen from the results in Table 7.6, the precision and recall of the static sliding-window approach, *Kasteren DBN* and *Patterson DBN*, are worse than those of EBAR in all cases but in NB for *Kasteren DBN*. Also, it shows that most of the instances created by the sliding-window approach are repetitive, showing a lower number of different consecutive instances. So, most of the created instances are repeated many times. This helped *Kasteren DBN* and *Patterson DBN* to obtain better results in terms of accuracy as it will be shown below. Comparing the configurations using the type of events as attribute, *EBAR*, and those without the events, *EBAR W-Ev*, it can be seen that the addition of the classified events as a new attribute improves the results. The improvements are more noticeable in the second dataset. However, in some cases like *Pat-EBAR DBN* the inclusion of this attribute does not affect the results.

If the number of instances is analyzed, a huge difference can be seen between the number of instances generated using static or dynamic windows. Using near 5% of the instances generated by the static-window approach, EBAR obtains much better results.

The experiments show that the right selection of the classification algorithm is important since the results obtained can be very different. DT is the best algorithm and NB obtains the worst results.

A deeper analysis can be seen in Table 7.7. It shows the precision of each class obtained by the best algorithm of the first dataset (DT), the instances or temporal windows created for each class and the real number of instances of each activity. The real number of instances is calculated counting the number of times the user is performing each activity in the dataset. For example, for the activity *Leaving* EBAR generates few instances, 353, since 353 significant events where found. Instead, the sliding-window creates a huge amount of them, 22582. Both methods are far from the real number of instances, 33. The activity *Leaving* activates just few sensors, since it represents the activity performed by the user when leaving the house. The sliding-window setup generates many instances even when the user is not activating any sensor. So, most of the instances generated during the time the user is out of the house are identical whereas the state-change approach does not create any. The same happens with *Idle* and *Sleeping*, where the accuracy is slightly better in *Kasteren DBN*. On the other hand, activities with a shorter duration such as *Breakfast* or *Drink* generate more instances using the state-change approach because more sensors are activated during the activity. For these shorter activities the precision of our models

Table 7.6: Models, precision, recall, number of instances and diversity of all configurations.

Models	Precision	Recall	N. Instances	Diversity
Kasteren DBN	80.55	80.08	40003	3.16%
Kas-EBAR DBN	86.77	86.11	2638	68.35%
Kas-EBAR PART	93.93	93.01	2638	68.35%
Kas-EBAR J48	92.89	92.07	2638	68.35%
Kas-EBAR HNB	91.05	91.11	2638	68.35%
Kas-EBAR NB	74.62	79.99	2638	68.35%
Kas-EBAR LB	92.79	91.70	2638	68.35%
Kas-EBAR DT	**94.59**	91.64	2638	68.35%
Kas-EBAR RF	91.75	91.27	2638	68.35%
Kas-EBAR kNN	83.04	81.65	2638	68.35%
Kas-EBAR SVM	93.17	91.89	2638	68.35%
Kas-EBAR W-Ev DBN	86.44	85.86	2638	68.35%
Kas-EBAR W-Ev PART	92.02	90.72	2638	68.35%
Kas-EBAR W-Ev J48	92.12	90.85	2638	68.35%
Kas-EBAR W-Ev HNB	90.10	89.27	2638	68.35%
Kas-EBAR W-Ev NB	74.35	79.66	2638	68.35%
Kas-EBAR W-Ev LB	93.12	91.35	2638	68.35%
Kas-EBAR W-Ev DT	92.87	90.59	2638	68.35%
Kas-EBAR W-Ev RF	91.50	91.04	2638	68.35%
Kas-EBAR W-Ev kNN	82.37	81.17	2638	68.35%
Kas-EBAR W-Ev SVM	92.94	91.72	2638	68.35%
Patterson DBN	78.90	86.57	16280	27.16%
Pat-EBAR DBN	93.67	96.92	5408	100%
Pat-EBAR PART	97.71	95.57	5408	100%
Pat-EBAR J48	98.37	94.83	5408	100%
Pat-EBAR HNB	99.22	97.74	5408	100%
Pat-EBAR NB	92.43	94.11	5408	100%
Pat-EBAR LB	97.56	98.42	5408	100%
Pat-EBAR DT	**99.34**	94.66	5408	100%
Pat-EBAR RF	96.05	97.3	5408	100%
Pat-EBAR kNN	99.30	98.15	5408	100%
Pat-EBAR SVM	97.01	97.4	5408	100%
Pat-EBAR W-Ev DBN	93.67	96.92	5408	100%
Pat-EBAR W-Ev PART	97.69	95.32	5408	100%
Pat-EBAR W-Ev J48	94.81	94.50	5408	100%
Pat-EBAR W-Ev HNB	97.97	96.51	5408	100%
Pat-EBAR W-Ev NB	93.80	92.42	5408	100%
Pat-EBAR W-Ev LB	97.54	98.42	5408	100%
Pat-EBAR W-Ev DT	**99.34**	94.66	5408	100%
Pat-EBAR W-Ev RF	96.05	97.3	5408	100%
Pat-EBAR W-Ev kNN	97.96	97.42	5408	100%
Pat-EBAR W-Ev SVM	96.51	97.40	5408	100%

is better. The results obtained in recall are very similar. So, our models obtain better precision and recall in average, because they recognize better activities that create fewer instances and they recognize just slightly worse the activities that generate a huge amount of instances using the sliding-windows method.

Table 7.7: Model, precision/number of generated instances, and real number of instances per class for the first dataset.

Model	Idle	Leaving	Toileting	Showering	Sleeping	Breakfast	Dinner	Drink
Kasteren DBN	92.5/4868	99.8/22582	58.9/218	89.2/223	99.8/11662	74.7/77	96.2/343	33.3/30
Kas-EBAR DT	86.4/540	97.7/353	86.5/610	100/131	96.7/377	97.9/243	98/264	93.5/119
Real N. Instances	86	33	80	23	23	20	10	19

Table 7.8 shows the precision of each class obtained by the best algorithm of the second dataset and the instances created for each class. This table shows that *Patterson DBN* creates more instances for all the activities than the other setup. However, the precision of *Pat-EBAR* is better in many classes. As in the first dataset, the activities with fewer instances created are better classified by our model and vice versa.

Table 7.8: Model, precision/number of generated instances, and real number of instances per class for the second dataset.

Model	Clear Table	Eat Breakfast	Front Door	Make Vanilla	Make Juice	Make Oatmeal
Patterson DBN	87.2/504	96.2/2731	43.8/296	95.5/4517	84.2/847	95.6/2626
Pat EBAR DT	100/367	98.9/666	100/42	96.3/1282	100/316	99.3/1064
Real N. Instances	10	17	10	51	11	22
Model	Make Eggs	Make Tea	Set Table	Use Bathroom	Use Phone	
Patterson DBN	83.7/1472	80.2/1532	87.9/715	36.1/422	77.5/618	
Pat EBAR DT	99.0/496	99.2/649	100/331	100/161	100/34	
Real N. Instances	32	32	11	10	10	

Also, EBAR reduces the features used in both datasets to just four: the last two sensors that generated the last two significant events, the previous activity and the event classification attribute. So, the models created by EBAR are quite simple, and the experiments show that they obtained better results.

7.4.4 Analysis of the Number of Created Windows

In this section, we briefly review how the results obtained with the first two datasets are affected by the number of generated windows. Hence, some configurations were tested generating models using the static sliding-window method for segmentation. In those configurations the parameters l and r, the length and the shift of the sliding-window, were changed in order to generate more instances. We are going to change one of the parameters in each dataset in order to show how the results change. In the next section, we are going to show how the results are affected changing l, the length of the window (Tables 7.11 and 7.12) using continuous data. In Table 7.9 the results obtained using the Kasteren dataset with l being 60, 30 and 15 seconds and $l = r$ are shown. The original configuration is marked in bold.

Table 7.10 shows the results obtained with the Patterson dataset using as values for r: 2, 1 and 0.5 seconds; and $l = 74$ seconds, the same ones used by Patterson. The original configuration is marked in bold.

Both tables show that as we decrease r, more instances are generated and better results are obtained. The number of generated instances also increases. The time intervals at which the activities are difficult to classify due to the sensor readings are always the same,

Table 7.9: Model and size of the window in seconds, precision, recall, number of instances generated, number of misclassified instances and percentage of significant events correctly classified using the Kasteren dataset.

Model	Precision	Recall	N. Instances	Misclassified I.	S.Events
DBN-60	80.55	80.08	40003	599	11.73%
DBN-30	89.71	89.97	80005	647	2.5%
DBN-15	94.32	94.73	160009	609	0.2%

Table 7.10: Model and shift of the window in seconds, precision, recall, number of instances generated, number of misclassified instances and percentage of significant events correctly classified using the Patterson dataset.

Model	Precision	Recall	N. Instances	Misclassified I.	S.Events
DBN-2	76.33	82.43	8140	1389	18.98%
DBN-1	78.9	86.57	16280	2390	13.42%
DBN-0.5	80.91	89.26	32560	4493	10.18%

mostly at the beginning of the activities or when sensors fail. However, the number of instances generated increases. So, most of the new instances generated decreasing l and r belong to periods of time at which nothing happens and are mostly correctly classified. The instances affected by those moments that are difficult to classify do not increase in the same proportion as the number of generated instances. So, for instance, in Table 7.9 the misclassified instances increase as we increase the number of generated instances, but not in the same proportion. Thus, the results improve. However, in the Kasteren dataset the number of misclassified instances does not change in the same way as in the Patterson dataset. The time instants at which the activities are difficult to classify affect to almost the same number of instances in the three configurations. This is due to the value of l which is much bigger than in the second dataset. So, the instances affected by the beginning of the activities are similar in all configurations. Hence, $l = 15$ generates a model, *DBN-15*, that is slightly better than the others, since it misclassifies less instances. As it generates many more instances, it also obtains better results in the metrics. Although increasing the number of instances improves the results, the classification of the significant events decrease as the number of instances augment. Table 7.11 shows the results obtained with the Huynh dataset using different windows lengths.

7.4.5 Classification Results Using Continuous Data

This section presents the results obtained using the third dataset. The results have been separated into two sections because the experiments use different types of data. In addition, in this set of experiments an analysis about how the number of created instances affects the results has been performed. For the experimental evaluation, the models described in Section 7.3.3 were generated. Precision and recall were used to evaluate the models using 10-fold cross-validation for estimating the error. Table 7.11 shows the size of the temporal window in seconds, the precision and recall and the instances created using the fixed-length sliding-window approach. The sizes of the windows are the sizes used by the authors of the dataset.

Table 7.11: Size of the windows, precision, recall and number of instances.

Size of W.	Precision	Recall	N. Instances
0.4	73.03	94.32	773817
0.8	69.71	92.54	386907
1.6	67.27	90.95	193453
2.4	65.92	89.17	128967
3.2	65.24	88.04	96725
4	65.68	88.46	77378

In the second experiment performed, the same ML algorithms described in 7.3.1 were used. Table 7.12 shows the best results using the algorithm DT that obtained the best result. Table 7.12 reports the threshold used to detect significant events, the precision and recall of the setups of the third dataset and the instances created using EBAR and the segmentation based on significant events. The thresholds that generate a number of instances similar to the ones on Table 7.11 are shown.

Table 7.12: Threshold, precision, recall and number of instances generated using EBAR for the Huynh dataset.

Threshold	Precision	Recall	N. Instances
<1	99.39	99.57	773817
1.0	98.83	98.54	203595
1.5	98.91	97.48	138444
2.0	98.54	96.49	121425
3.0	98.36	94.65	100748
5.0	97.63	95.34	76919

Both segmentation algorithms generate the same number of instances when the threshold and windows' size used are the lowest ones. In such cases, both segmentation methods are equivalent. However, the results obtained by EBAR are better. It can be seen that there is a correlation between the number of instances created and the results obtained in all but the last row of the two tables. So, generating more instances can be a good way to improve the results. Nevertheless, to increase around 2% of precision and 4% of recall, the number of instances has to be augmented more than 10 times. The best results are achieved also using DT. In this dataset, the classification of the significant events does not affect the result since, as it has been shown in Section 7.4.2, the classification of the significant events is not good.

7.4.6 Adapting EBAR to Improve the Results

The results offered in Section 7.4.4 could imply that the fixed-length sliding-window could behave better by modifying its parameters and improve its results by generating more instances. EBAR also permits to improve the results generating more instances. To do that, more significant events have to be used in the segmentation phase. One experiment was performed to show this behavior with the Patterson and the Huynh datasets. Table 7.12 in Section 7.4.5 shows that by lowering the value of the threshold, more instances are generated and the results are improved.

In order to test this behavior with the Patterson dataset, models were generated with EBAR using as significant events all the readings of the RFID. Table 7.13 shows the results obtained with the second dataset. The results obtained by the best and the worst algorithms are shown.

Table 7.13: Precision, recall, number of generated instances and percentage of significant events correctly classified using the Patterson dataset and all events.

Model	Precision	Recall	N. Instances	Significant events
Pat-EBAR kNN	99.80	99.45	23138	87.96%
Pat-EBAR NB	97.80	99.21	23138	34.72%

Section 7.4.5 shows that by increasing the number of generated instances, the results improve. On the other hand, comparing the results of Table 7.13 and Table 7.6 it can be seen that both algorithms improve their performance. Also, Table 7.13 shows that the classification rate of the significant events slightly varies, obtaining better results in the worst algorithm and worsening the best results less than 2%. So, the experiment shows that augmenting the number of generated instances improves the overall results. But, the classification rate of significant events does not vary significantly or even worsens.

The segmentation used to generate the results of Table 7.13 is the same as was used in [Modayil *et al.*, 2008]. In order to show the performance with respect to this work the simple HMM has been replicated, through an equivalent DBN, used by Modayil *et al.* and a precision of 74.96 and a recall of 85.54 were obtained, which are quite similar to the results obtained by the Patterson DBN in Table 7.6 but far from the results presented in Table 7.13.

7.5 Discussion

This chapter presented a new algorithm for human activity recognition from sensor data. It uses the events produced by sensors to create the instances for classification, instead of using fixed-length sliding-windows. From those events a new feature is generated performing a previous classification to determine if the detected events enclose an activity. This permits EBAR to improve the performance over other approaches.

It has been shown in Section 2.1.2 that other approaches also use events for segmentation and, in some cases, the segmentation can be equivalent depending on the significant events used by our models. Although others have used events before, the main difference with our approach lies on how those events are selected. Although any event could be used, we try to find the boundaries of the activities to detect when the user switches from an activity to another in order to detect how the activities change the environment. Hence, the AR system can be used to automatically generate a model of the activities that the user performs. This model could be used to replicate the behavior of the user in order to provide assistance or even predict the next action through the generation of sequences of activities to reach a goal. Such a system will be presented in the next chapter.

While other approaches select the events only to obtain better recognition rates, our method tries to obtain better recognition rates but, at the same time, EBAR detects the boundaries of the activities to correctly generate action models able to replicate the user activities. In Section 7.4.1, it is shown that the correct selection of significant events obtains better results recognizing the instances where the boundaries of the activities are. This is the key aspect of EBAR, since bad recognition rates of these instances do not permit to correctly learn how each activity modifies the environment. It would cause part of the changes in the environment produced by one activity to be assigned to another activity.

Good performance in public datasets used in the past by other researchers has been shown. EBAR obtains better precision and recall than other approaches in average. Although other approaches may obtain better results in some activities, the main advantage of our algorithm is that it can provide reasonably accurate models for all the activities and obtain better recognition rates in the boundaries of the activities.

It has been seen that augmenting the number of instances may improve the results. However, the recognition rates of the instances in the boundaries of the activities may decrease using the fixed-length sliding-window or does not vary significantly employing EBAR. In any case, this may not be feasible for some applications because it also augments the computational cost. For instance, in [Amft *et al.*, 2007; Hoey *et al.*, 2011] their authors present distributed systems in sensor networks where each node contributes to the activity recognition by processing its data and recognizing some activities. So, in applications where the data is processed with low computational power, generating too many instances may prevent the system from working. In addition, generating fewer instances may help to save battery consumption.

In addition, our approach is independent of the time granularity since it uses events to build temporal windows in contrast to the sliding-window approach where the user usually has to modify the windows length according to the activities temporal duration.

Chapter 8

Generating Planning Action Models

Automated planning has been successfully used in many domains like robotics or transportation logistics. However, building an action model is a difficult and time-consuming task even for domain experts. This chapter presents a system, ASRA-AMLA, for automatically generating planning action models from sensor readings. Activity recognition is used to extract the actions that a user performs and the states produced by those actions. Then, the sequences of actions and states are used to infer a planning action model. In this chapter, we build on the previous work presented in Chapter 7 where the EBAR system for AR was presented. With this approach, the system can automatically build an action model related to human-centered activities. It allows us to automatically build an assistance system for guiding humans to complete a task using Automated Planning. To test our approach, a new dataset from a kitchen domain has been generated. The tests performed show that our system is capable of correctly extracting actions and states from sensor time series and creating a planning domain used to guide a human to successfully complete a task.

8.1 Introduction

Activity recognition (AR) systems have been widely used in the past to detect activities of daily living (ADL) [Lawton and Brody, 1969], but most of the literature describes approaches that detect the whole activity like in [Patterson *et al.*, 2005; Logan *et al.*, 2007; Kasteren *et al.*, 2008; Kasteren *et al.*, 2011; Dernbach *et al.*, 2012]. In order to provide assistance for the user to complete an activity or task, it would be desirable to detect the subtasks or actions (e.g., opening/closing cabinets) that compose an activity (e.g., preparing an omelette) and the effects that these actions have (omelette cooked). So, recognizing such actions and their effects while a user tries to accomplish a task could be used to check if the user is completing the task correctly. This way, a correct sequence of actions can be generated for a user to accomplish the task successfully. With the purpose of generating the sequence of actions, Automated Planning (AP) tools could be used, but they require having an action model. Such model could be generated by experts manually, but usually this is a time-consuming and error-prone process.

Hence, the goal of the work presented in this chapter is to build a system capable of guiding users while they are completing an activity. The system will be able to recognize, from sensor time series, the actions performed by a user, and the states of the system produced by those actions. Using that information, the proposed system generates a user action model represented as a STRIPS (Stanford Research Institute Problem Solver) [Fikes and Nilsson, 1971] planning domain in the standard language PDDL (Planning Domain Definition Language) [Mcdermott, 2000]. Then, the generated planning domain will be used by an automated planner to generate plans (sequences of actions) to accomplish a task. These plans can be used to provide assistance to people in daily activities.

This chapter describes a system, called ASRA-AMLA(Action and States Recognition Algorithm - Action Model Learning Algorithm), that automatically learns a user action model to assist users while they cook a recipe. Also, this action model could be used for plan recognition, as it is shown in [Ramírez and Geffner, 2009]. The proposed system goes beyond other works presented in the literature like [Yang *et al.*, 2005; Mourao *et al.*, 2009] on the automatic generation of planning domains since it is capable of learning the AP domain using traces of information from sensors instead of traces provided by humans.

In order to automatically generate the planning action model, the preconditions and effects of each action have to be learned from sensor readings. For that reason, the segmentation of the time series is a key issue since the typical method used, the temporal sliding-windows, may overlap several actions. Instead, this chapter describes work that employs the method based on events described in Chapter 6. The events produced by changes in the environment, like the actions do, are used to split the sensor time series. A segmentation method based on events may produce better results for generating planning action models than a method based on fixed-length sliding-windows as it was shown in Chapter 6. Once the sensors time series are segmented, two different models to recognize actions are compared employing different features in order to obtain the best classifier. We used ten different machine learning algorithms in the experiments. The best model is then used to generate the sequences of actions employed to build the action model in PDDL.

8.2 ASRA-AMLA System Description

ASRA-AMLA is composed of two modules. The first one, ASRA (Action and States Recognition Algorithm), extracts actions and states from the sensor readings, creating a sequence of interleaved actions and states. The second one, AMLA (Action Model Learning Algorithm), builds an AP domain from the sequences generated by ASRA.

The system has three modes of operation. Using the first mode, called *Phase 1*, ASRA learns the classifier that will be used to classify new sensor data. Figure 8.1 shows how the system works in this mode.

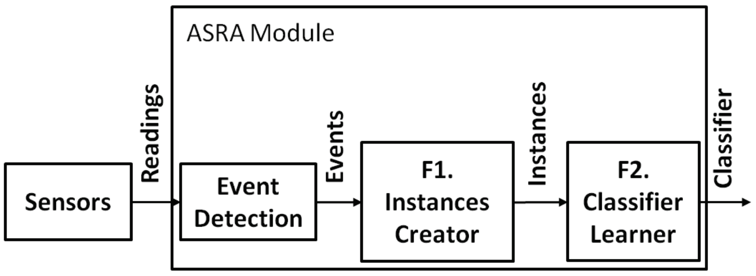

Figure 8.1: Phase 1 working mode of ASRA-AMLA.

Once the action classifier has been build, ASRA-AMLA is used to build planning domains.

This is the second working mode of the system and is called *Phase 2*. Figure 8.2 shows a schema of ASRA-AMLA working in this mode.

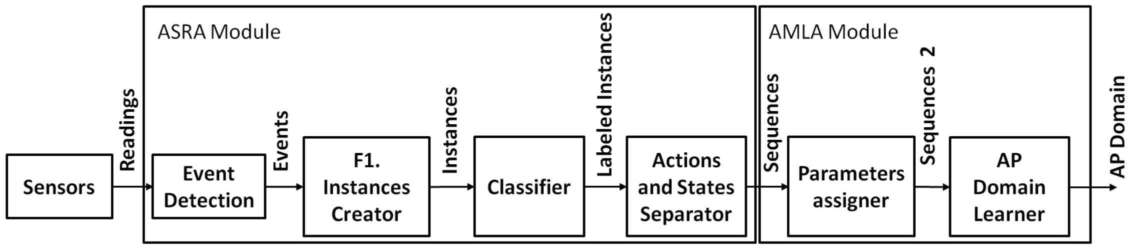

Figure 8.2: Phase 2 working mode of ASRA-AMLA.

The third working mode is the *Run-time* mode. After learning an AP domain, AMLA creates AP problems using the states provided by ASRA and, using the AP problems and the AP domain, executes a planner to generate plans. Figure 8.3 shows how ASRA-AMLA generates plans to guide the users through the actions that compose the task. Both modules are described in the next sections.

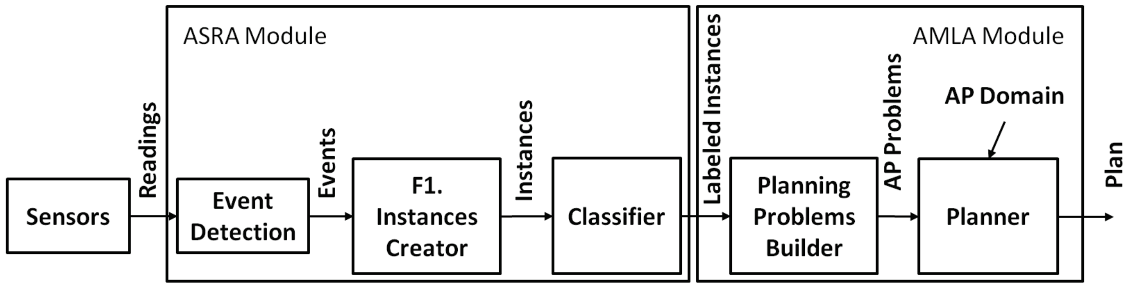

Figure 8.3: Run-time mode of ASRA-AMLA.

8.2.1 ASRA: Action and States Recognition Algorithm

The objective of ASRA-AMLA is to learn a planning action model, an AP domain in PDDL. So, in order to accomplish such a task, the ASRA module recognizes the actions the user performs and also provides enough information to learn the preconditions and effects of each action recognized by the system. It recognizes the state of the environment before and after every action. In order to recognize states, the sensors should detect the changes that the actions produce. For instance, an accelerometer in an arm can provide information to recognize certain activities [Pham and Olivier, 2009] (e.g., peeling), but it does not provide information about the effects of those activities (e.g., potato1 is peeled). In our setup three types of sensors are used: magnetic sensors, RFID's and cameras. Those sensors are going to gather information while the user cooks a recipe in a kitchen where the sensors have been installed. The magnetic sensors provide information about the state of the furniture of the kitchen; this way, they capture the effects that the actions of the user have on the furniture. The RFID's are used to detect the objects that the user is using. The cameras are used to detect the location and state of the objects and appliances. Thus, the cameras detect the effects that the actions of the user cause on the objects. For instance, when the user opens a drawer to pick up a fork and leaves it on the kitchen top, the magnetic sensors detect that the drawer has been opened and closed, RFIDs detect that the fork has been picked up and put down, and the cameras detect that the fork has been left on the kitchen top.

Once the sensors are able to detect the effects of the actions through the states, the ASRA module defines what an event is to be used for segmentation. The segmentation method based on events presented in Chapter 6 is used.

When the user performs an action, it produces some changes in the environment and those changes may generate none or several changes in the sensor readings. For example, when a user picks up an object, the object changes its location. Thus, an RFID sensor placed near the hand of the user may change its reading from not detecting anything to detecting the object. We are interested on detecting those kinds of changes in the sensor readings since they are connected with the effects of the action *pick-up*. An *event* is defined as any change in the readings of any sensor. For example, when the user opens a cabinet, an RFID detects the cabinet and generates an event. Also, the magnetic sensor changes its value from closed to open.

The events are used for the segmentation of the sensor time series to extract the actions and states. When an event is detected, the system recognizes the action to which it belongs and its effects. If the action generates more than one effect, it may also generate more than one event. Also, one event may be generated by more than one action, even for actions we are not interested in.

For magnetic sensors or RFID's, the events are easy to define since they provide discrete values. For both sensors, all the changes in the values of the readings they report will be considered as an event. For cameras, the detection of the events is not so direct since they provide richer information, and such information may not be discrete. They are used to detect the state of some appliances (on and off) and objects (e.g., cracked for an egg) and the location of some objects (e.g., fry-pan on the burner). So, cameras will report in every frame the state of the appliances that they are monitoring, the objects that they can detect, and the state of those objects. Hence, the changes in the location or state of any object detected by the cameras are considered as events.

Once the events have been defined, the next step is to build a classifier that recognizes the action the user has performed. The action recognition task is formally defined in Section 6.2.

After learning the action classifier in *Phase 1*, the *Phase 2* is used to learn planning domains. In *Phase 2* the classifier is used to classify the instances. Then the states recognition is performed. When activities are recognized, sensors readings are segmented in pieces of data called temporal windows. These temporal windows do not usually match a complete activity from the beginning to the end of the activity. Instead, they normally split an activity into pieces that contain data that belongs to some parts of the activity. Actions have the same problem. Thus, in order to recognize complete actions, we assign to each temporal window an action. Next, we group consecutive temporal windows classified with the same action. Then, ASRA extracts the states between two different actions.

Algorithm 3 shows how ASRA works in *Phase 2* mode. S contains the states and A the actions that the module extracts. *CurrentState*(*cont*) obtains the state given by the parameter *cont*. It returns the state of the system given by the sensor readings after *cont* events. Notice that the system is working obtaining the sensor readings from a log file. *EventDetector*() is a function that returns the next event of the sensors time series, $FeaturesExtraction(e, s, S)$ extracts the features from the sensor readings after the event

Algorithm 3 Action and States Recognition Algorithm (ASRA).

```
Input: $logFile$
Output: $StatesandActionssequence$

  $A \leftarrow null$
  $a' \leftarrow null$
  $cont \leftarrow 0$
  $S \leftarrow CurrentState(cont)$
  $s \leftarrow null$
  $e \leftarrow EventDetector()$
  while $e \neq null$ do
    $s \leftarrow CurrentState(cont)$
    $i \leftarrow FeaturesExtraction(e, s, S)$
    $a \leftarrow ActionClassifier(i)$
    if $(a \neq a')$ and $(a' \neq null)$ then
      $S \leftarrow S + s$
      $A \leftarrow A + a'$
    end if
    $cont \leftarrow cont + 1$
    $a' \leftarrow a$
    $e \leftarrow EventDetector()$
  end while
  $S \leftarrow S + s$
  $A \leftarrow A + a'$
  $sequence \leftarrow Merge(A, S)$
  return $sequence$
```

e contained in s and the sensor readings before e contained in S and creates an instance that is classified by $ActionClassifier()$. $ActionClassifier()$ is the function built by f_2 in *Phase 1* mode. It uses the algorithm EBAR presented in Chapter 7. For each event, $ActionClassifier$ returns a class (action). If the class is different than the previous one, the past action and state are saved in A and S. Finally, A and S are merged to generate a sequence of interleaved states and actions: $(state - action)^* - state$.

As an example, suppose the user performs three consecutive actions, a, b and c, that generate the following sequence of events detected by the system $e_1 - e_2 - e_3 - e_4 - e_5 - e_6$ where a generates the events e_1 and e_2, b generates e_3, e_4, and e_5 and c generates e_6. If the initial state is s_I, the sequence of actions and states input to the action classifier is $s_I - e_1 - s_1 - e_2 - s_2 - e_3 - s_3 - e_4 - s_4 - e_5 - s_5 - e_6 - s_F$ and the events could be classified as $s_I - a_1 - s_1 - a_2 - s_2 - b_3 - s_3 - b_4 - s_4 - b_5 - s_5 - c_6 - s_F$. Then, consecutive events classified as belonging to the same action are grouped, generating the sequence $s_I - a - s_2 - b - s_5 - c - s_F$. This sequence is the input for the AMLA module.

Next, we are going to show an example about how the system works using real data. First, the sensor data is segmented using events. Table 8.1 shows an example with two events and the states that they produce. The first event is produced when the user opens *drawer1*. The second event is produced when the user picks up *fork1*.

Then, a change of representation is performed. AP domains generally use predicate

Table 8.1: Sensor data segmentation.

rfid rfid1 null		rfid rfid1 null		rfid rfid1 null
rfid rfid2 null		rfid rfid2 null		rfid rfid2 fork1
reed drawer1 closed		reed drawer1 opened		reed drawer1 opened
reed fridge closed		reed fridge closed		reed fridge closed
reed dcup1 closed	event 1	reed dcup1 closed	event 2	reed dcup1 closed
reed dcup2 closed		reed dcup2 closed		reed dcup2 closed
reed ucup1 closed		reed ucup1 closed		reed ucup1 closed
reed ucup2 closed		reed ucup2 closed		reed ucup2 closed
cameraState cooktop off		cameraState cooktop off		cameraState cooktop off

logic as representation paradigm, so the states obtained from sensor readings have to be translated into predicates. We have created a mapping from low-level features (sensor readings) to predicates. This mapping is applied to sensor readings to automatically obtain corresponding predicate formulae. The information that sensors provide is related to the location and the state of the objects (e.g., opened cupboard1, fry-pan on the burner). The state of the objects is represented with a predicate with the same name as the fact in the state and one parameter, the object. For example: *(opened cupboard1)*, *(beaten egg1)*. For the location of objects we use the predicate *(in param1 param2)* where *param1* is the place where the object *param2* is. RFIDs are used to detect the objects that the user is holding, so they generate the predicate *(holding obj_n)* when an RFID detects the object obj_n.

Table 8.2: Mapping from sensor readings into predicates.

rfid rfidId object	(holding object)
reed furniture opened	(opened container) (closed container)
cameraState object state	(on object) (off object) (raw object) (cracked object) (beaten object) (omelette object)
cameraPlace object location	(in location object) (inside recipient ingredient)

Table 8.2 shows how the system translates sensor readings into predicates. RFID sensors generate the reading *rfid rfidId object* where *rfidId* is the identifier of the RFID and *object* is the object detected. These sensors produce the predicate *(holding object)* where *object* is the object that the user is holding. In states in which no objects are detected by the RFIDs, no *(holding object)* predicates are included. *reed furniture state* shows the state of the cupboards, drawer and fridge indicated by the magnetic sensor in each piece of *furniture*. The states can be *opened* or *closed*. Cameras generate two types of readings: *cameraState object state* and *cameraPlace object location*. *cameraState object state* indicates the state

Table 8.3: Example of translation from sensor readings into predicates.

rfid rfid1 null	-
rfid rfid2 fork1	(holding fork1)
reed drawer1 opened	(opened drawer1)
reed fridge closed	(closed fridge)
reed dcup1 closed	(closed dcup1)
reed dcup2 closed	(closed dcup2)
reed ucup1 closed	(closed ucup1)
reed ucup2 closed	(closed ucup2)
cameraState cooktop off	(off burner)

of the objects detected by the cameras. The appliances can be in the states *on* or *off* and the eggs can be in the states *raw*, *cracked*, *beaten* or *omelette* when they are fried. There is no other object that changes its state in our experiments. *cameraPlace object location* can generate two predicates. Both predicates indicate the location of the *object*. The difference between both predicates is that the predicate *in* is used to indicate a location of the kitchen and the predicate *inside* is used to indicate the location of an object inside another object.

Table 8.3 shows an example of how the system translates the sensor readings of a state into predicates. In the example, the user is holding the *fork1* and the *drawer1* is opened. The rest of the state is extracted from the initial state. In the beginning, the objects are located inside the cupboards, drawer or fridge and the sensors are not capable of detecting them. For that reason, an initial state is used to specify the location of the objects and it is used to complete the current state. In the example, *rfid2* is detecting *fork1* and *rfid1* is not detecting anything. *drawer1* is *opened* and the rest of furniture is *closed*. The cameras detect that the *cooktop* is *off*.

Table 8.4: Example of sequence with three states and two events produced by one action.

(holding fork1)		-		(in kitchentop fork1)
(opened drawer1)		(opened drawer1)		(opened drawer1)
(closed fridge)		(closed fridge)		(closed fridge)
(closed dcup1)		(closed dcup1)		(closed dcup1)
(closed dcup2)		(closed dcup2)		(closed dcup2)
(closed ucup1)		(closed ucup1)		(closed ucup1)
(closed ucup2)	(put-down fork1 kitchentop)	(closed ucup2)	(put-down fork1 kitchentop)	(closed ucup2)
(off burner)	RFID Event	(off burner)	Camera Event	(off burner)
(in frypan dcup1)		(in frypan dcup1)		(in frypan dcup1)
(in plate1 ucup2)		(in plate1 ucup2)		(in plate1 ucup2)
(in bigbowl ucup1)		(in bigbowl ucup1)		(in bigbowl ucup1)
(in oilbottle dcup2)		(in oilbottle dcup2)		(in oilbottle dcup2)
(in egg1 fridge)		(in egg1 fridge)		(in egg1 fridge)
(inside oil oilbottle)		(inside oil oilbottle)		(inside oil oilbottle)
(raw egg1)		(raw egg1)		(raw egg1)

After performing the mapping, the segmented sensor data is composed of sequences of events and states described using predicates. Table 8.4 shows an example of a sequence composed of three states and two events produced by one action. The example shows a user holding a fork in the first state. Then, the user puts the fork down and it is detected on the kitchen top. The first event is produced when the RFID stops detecting the fork. The second event is produced when the cameras detect the object on the kitchen top.

Table 8.5: Pre-operator structures.

put-down	put-down
ADDS:	ADDS:
DELS:	(in kitchentop fork1)
(holding fork1)	DELS:
PRECONDITIONS:	PRECONDITIONS:
(holding fork1)	(opened drawer1)
(opened drawer1)	(closed fridge)
(closed fridge)	(closed dcup1)
(closed dcup1)	(closed dcup2)
(closed dcup2)	(closed ucup1)
(closed ucup1)	(closed ucup2)
(closed ucup2)	(off burner)
(off burner)	(in frypan dcup1)
(in frypan dcup1)	(in plate1 ucup2)
(in plate1 ucup2)	(in bigbowl ucup1)
(in bigbowl ucup1)	(in oilbottle dcup2)
(in oilbottle dcup2)	(in egg1 fridge)
(in egg1 fridge)	(inside oil oilbottle)
(inside oil oilbottle)	(raw egg1)
(raw egg1)	

When the first event occurs, the predicate *(holding fork1)* disappears. After the second event, the predicate *(in kitchentop fork1)* appears. So, when one event occurs, we know the predicates whose value remains equal (true or false), the predicates that appear (their values were false and they are true) and the predicates that disappear (their values were true and they are false). Actions effects are learned using the difference between the state before the action s_b and the state after it s_a. Thus, the *adds* list of the operator a is: $add(a) = s_a \setminus s_b$ (set difference of state after a and state before a) and the *deletes* list is: $del(a) = s_b \setminus s_a$. Before learning the final operators we used what we called *pre-operator* which contains the adds, deletes and preconditions that will be used to build the final operators. Table 8.5 shows an example of two *pre-operators* used to save two events produced by the action *put-down*. They are also used to build the learning instances to classify the actions that the user performs. Each structure is composed of three parts. The first one, *ADDS*, contains the *adds* list. The second one, *DELS*, contains the *deletes* list. The third part, *PRECONDITIONS*, contains the predicates that compose s_b.

Table 8.6: Pre-operator structure after merging actions.

put-down
ADDS:
(in kitchentop fork1)
DELS:
(holding fork1)
PRECONDITIONS:
(holding fork1)
(opened drawer1)
(closed fridge)
(closed dcup1)
(closed dcup2)
(closed ucup1)
(closed ucup2)
(off burner)
(in frypan dcup1)
(in plate1 ucup2)
(in bigbowl ucup1)
(in oilbottle dcup2)
(in egg1 fridge)
(inside oil oilbottle)
(raw egg1)

When the actions classifier has to be built, in *Phase 1*, the action that generates these structures is also included. In the example, *action* would be *put-down* in both events. In *Phase 2* or in *Run-time*, when the classifier is already available and the actions have to be classified, *action* is substituted by the action provided by the classifier.

After the classification, consecutive events classified as belonging to the same action are grouped. Then, when two or more events have been classified as the same action, in this case *put-down*, the two pre-operators are merged in just one as Figure 8.6 shows. The way the pre-operators are merged is joining the *ADDS* and *DELS* and taking the *PRECONDITIONS* of the first pre-operator. Thus, given two consecutive pre-operators p_n and p_{n+1} generated by the events n and $n+1$ and classified as the same action a, then a new pre-operator p' is generated merging p_n and p_{n+1} where $ADDS_{p'} = ADDS_{p_n} \cup ADDS_{p_{n+1}}$, $DELS_{p'} = DELS_{p_n} \cup DELS_{p_{n+1}}$ and $PRECON_{p'} = PRECON_{p_n}$. Then, p_n and p_{n+1} are substituted by p'. After this step, the sequences are ready to be used by the AMLA module.

8.2.2 AMLA: Action Model Learning Algorithm

This module builds an AP domain from the sequences generated by the ASRA module. These sequences are traces of a user performing an activity in the form of *pre-operators*.

Table 8.7: Types, predicates and actions used.

types	staticobject moveable - object surface furniture - staticobject appliance worktop - surface ingredient utensil - moveable container normalutensil - utensil cookingcontainer normalcontainer - container specialcontainer - normalcontainer liquidcontainer - normalcontainer
predicates	(raw ?x - ingredient) (in ?x - moveable ?y - staticobject) (closed ?x - staticobject) (opened ?x - staticobject) (holding ?x - object) (inside ?x - ingredient ?y - container) (on ?x - appliance) (off ?x - appliance) (raw ?x - ingredient) (cracked ?x - ingredient) (beaten ?x - ingredient) (omelette ?x - ingredient)
actions	(pick-up ?x - moveable ?y - surface) (put-down ?x - moveable ?y - surface) (get-out ?x - moveable ?y - furniture) (put-away ?x - moveable ?y - furniture) (open ?x - furniture) (close ?x - furniture) (switch-on ?x - appliance) (switch-off ?x - appliance) (transfer ?x - ingredient ?y - container ?z - container) (fry ?x - ingredient ?y - cookingcontainer ?z - appliance) (beat ?x - ingredient ?y - normalutensil ?z - container) (crack-egg ?x - egg ?y - specialcontainer)

AMLA is given the following inputs for learning: (1) object types and generic predicates defined manually, as in Table 8.7; (2) state-action sequences automatically generated by ASRA and (3) the type of the objects involved in each action (parameters of the actions) defined manually. Table 8.7 shows the object types, generic predicates, and actions defined.

Also, some standard assumptions are made. First, it is assumed that actions are deterministic. Thus, actions are going to have always the same effects. When the user performs an action, the sensors' noise or unexpected effects affect the system mainly when the AP domain is learned. Once a domain has been learned, in *Run-time*, the sensors' noise affects the system only when it is going to generate the initial state of the AP problem. A non-deterministic domain could be used to incorporate into the model the unexpected effects of some of the actions or the noise of the sensors but these domains are more complex. Instead, we have used a deterministic domain because it is capable of providing guidance for the user to reach his/her goal, which is one of the goals of this thesis, even when they do not model the noise of the system.

Second, it is assumed that the preconditions are conjunctive; all the preconditions have to be true for the action to be executed. Also, a threshold (error rate) is used to eliminate the sensors' noise. This threshold indicates a minimum percentage of times that a predicate has to appear before/after an action to be used for building the corresponding planning operator. The goal is to learn the preconditions and effects (adds and deletes) of the actions. This threshold allows us to deal with situations in which the action fails (i.e., the effects are not correct); or the sensors fail (i.e., the preconditions or effects are not correct). In the experiments, we have varied the values of the threshold in order to evaluate its impact.

Algorithm 4 Algorithm for parameters assignment.

```
Input: $operatorParametersTypes$ , $pre - operator$, $ontology$
Output: $parameters$

  $parameters \leftarrow \emptyset$
  $adds \leftarrow ExtractParameters(pre - operator.adds)$
  $dels \leftarrow ExtractParameters(pre - operator.dels)$
  $possibleParameters \leftarrow adds \cup dels$
  for each $parameter$ in $operatorParametersTypes$ do
    $index \leftarrow 0$
    $object \leftarrow ElementAt(possibleParameters, index)$
    $type \leftarrow TypeOf(object, ontology)$
    while $(parameter \neq type)$ and $(index \leq (size(possibleParameters) - 1))$
    do
      $index \leftarrow index + 1$
      $object \leftarrow ElementAt(possibleParameters, index)$
      $type \leftarrow TypeOf(object, ontology)$
    end while
    if $(parameter = type)$ then
      $parameters \leftarrow parameters + object$
    else
      $parameters \leftarrow parameters + "unknown"$
    end if
  end for
  return $parameters$
```

Before learning the operators, the parameters have to be provided for each action. These parameters are the arguments given to each planning operator. In this case, they are the objects involved in each action performed by the user. In the last step of ASRA, the actions are merged and the effects are joined. In order to provide the parameters, the *ADDS* and *DELS* of the pre-operators are used. Then, among the parameters of the predicates that are contained in the *ADDS* and *DELS* of each pre-operator, those that are of the same type that the parameters of the action that was assigned to the pre-operator are used. If none of the parameters of the predicates that are contained in the *ADDS* and *DELS* belong to the type of a parameter of the action, then the parameter is assigned as *unknown*.

Algorithm 4 shows how the parameters are assigned to a pre-operator. First, all objects are extracted from the *ADDS* and *DELS* using the function *ExtractParameters*. Next, the algorithm goes over each of the parameters of the operator to which the pre-operator belongs. The parameters of each operator are defined by the user and are provided to the algorithm. Then, the algorithm searches an object in *possibleParameters* with the same type as the parameter of the operator. *ElementAt* returns the object in the position given by *index* and $TypeOf$ returns the type of *object* indicated in *ontology*. *ontology* is the definition of the object types hierarchy. Then, *type* is compared with the type of the parameter of the planning operator.

Table 8.8: Pre-operator structure with parameters.

put-down fork1 kitchentop
ADDS:
(in kitchentop fork1)
DELS:
(holding fork1)
PRECONDITIONS:
(holding fork1)
(opened drawer1)
(closed fridge)
(closed dcup1)
(closed dcup2)
(closed ucup1)
(closed ucup2)
(off burner)
(in frypan dcup1)
(in plate1 ucup2)
(in bigbowl ucup1)
(in oilbottle dcup2)
(in egg1 fridge)
(inside oil oilbottle)
(raw egg1)

Keeping on with the previous example shown in Table 8.6, Table 8.8 shows a pre-operator with its parameters. The *ADDS* contain the predicate *(in kitchentop fork1)* and the *DELS* contain the predicate *(holding fork1)*. Then, the possible parameters would be *kitchentop* and *fork1*. In Table 8.7, we can see that the action *put-down* has as parameters *x? moveable ?y - surface*. *kitchentop* is a *surface* and *fork1* is a *moveable*. Then, the final name of the action would be *put-down fork1 kitchentop*. If the fork would have not been detected on the kitchen top, the predicate *(in kitchentop fork1)* would not be in the pre-operator. In such case, the action and parameters would be *put-down fork1 unknown*.

Algorithm 5 Algorithm for learning preconditions.

```
Input: $pre-operators$ , $threshold$
Output: $preconditions$

  $predicates \leftarrow \emptyset$
  $preconditions \leftarrow \emptyset$
  for each $pre-operator$ in $pre-operators$ do
    for each $predicate$ in $pre-operator.preconditions$ do
      $counter \leftarrow 0$
      $parameters \leftarrow ExtractParams(predicate)$
      for each $parameter$ in $parameters$ do
        if $parameter \in pre-operator.params$ then
          $counter \leftarrow counter + 1$
        end if
      end for
      if $counter > 0$ then
        $predicate' \leftarrow ChangeParameter(predicate, pre-operator)$
        $predicates \leftarrow predicates + predicate'$
      end if
    end for
  end for
  for each $predicate$ in $predicates$ do
    $counter \leftarrow Count(predicate, predicates)$
    $counterTotal \leftarrow Size(pre-operators)$
    $limit \leftarrow Percentage(counterTotal, threshold)$
    if $counter \geq limit$ then
      $preconditions \leftarrow preconditions + predicate$
    end if
  end for
  return $preconditions$
```

After assigning the parameters to each pre-operator, the domain is learned. All the pre-operators that have been classified as the same operator are put together. Then, for each of these pre-operators Algorithm 5 is executed to learn the preconditions. For each pre-operator, the predicates where the parameters of the pre-operator are not present are removed. So, $pre-operator.preconditions$ are the predicates that compose the preconditions of a given pre-operator. $ExtractParams(predicate)$ returns the objects that are in the parameters of *predicate*. The algorithm verifies if each parameter of the predicate is in the parameters of the pre-operator $pre-operator.params$. If at least one of the parameters of the predicate is part of the parameters of the pre-operator, the predicate is saved. Before being saved, the predicate is modified by $ChangeParameter(predicate, pre-operator)$ as follows: given the pre-operator $preo(objectA, objectB, objectC)$ and the predicate $pred(objectB, objectD)$, the predicate is modified into $pred(param2, objectD)$ since $objectB$ is the second parameter of $preo$. The predicate $pred2(objectA, objectC)$ would be modified into $pred2(param1, param3)$.

Once all the predicates, where at least one of the parameters of the pre-operator is present, have been saved, these predicates are counted: $Count(predicate, predicates)$. Then, the threshold indicates the minimum percentage of times that a predicate must appear to be part of the preconditions of the final planning operator. So, the number of times that a predicate must appear is calculated by the function $Percentage(counterTotal, threshold)$ given the total number of pre-operators $Size(pre-operators)$.

For example, suppose we want to learn the operator a, which has 10 instances in the input sequence, and we had set a threshold of 70%. Any predicate containing any parameter of the action, which is present in the previous state of the action a at least 7 out of 10 times would be included as a precondition for that action.

The threshold is also used for the effects in the same way as for the preconditions. To be part of the effects, a predicate has to appear in the effects a percentage of times greater than or equal to the threshold. Algorithm 6 shows how the *ADDS* are learned. *DELETES* are learned identically. The only difference is that none of the predicates that are part of the *ADDS* are removed before comparing the number of times they appear and the threshold.

Algorithm 6 Algorithm for learning effects.

```
Input: $pre-operators$ , $threshold$
Output: $adds$

  $predicates \leftarrow \emptyset$
  $adds \leftarrow \emptyset$
  for each $pre-operator$ in $pre-operators$ do
    for each $predicate$ in $pre-operator.adds$ do
      $predicate' \leftarrow ChangeParameter(predicate, pre-operator)$
      $predicates \leftarrow predicates + predicate'$
    end for
  end for
  for each $predicate$ in $predicates$ do
    $counter \leftarrow Count(predicate, predicates)$
    $counterTotal \leftarrow Size(pre-operators)$
    $limit \leftarrow Percentage(counterTotal, threshold)$
    if $counter \geq limit$ then
      $adds \leftarrow adds + predicate$
    end if
  end for
  return $adds$
```

Once a domain has been learned, ASRA-AMLA is used in *Run-time* mode 8.3. This working mode builds planning problems using the current state given by the sensors. So, when an event is detected, a learning instance is built and the action that generated the event is classified. Then, the values of the sensors after the event (the current state) along with the goal state (omelette cooked) are used to generate a planning problem. Finally, the planner is executed using the new planning problem and the generated planning domain to generate a sequence of actions that the user should execute to achieve his/her goal. That way, the system provides assistance to the users generating a plan from every single state. Algorithm 7 describes how the system works in *Run-time* mode. Notice that the classification of the actions is not needed but it is included to let the system monitor the user actions at the same time that it provides assistance generating plans.

Algorithm 7 Run-time Algorithm.

```
Input: $logFile$, $domain$, $goalState$
Output:

  $cont \leftarrow 0$
  $s \leftarrow CurrentState(cont)$
  $e \leftarrow EventDetector()$
  while $e \neq null$ do
    $i \leftarrow FeaturesExtraction(e)$
    $a \leftarrow ActionClassifier(i)$
    $problem \leftarrow ProblemGenerator(s, goalState)$
    $plan \leftarrow planner(problem, domain)$
    $show(plan)$
    $cont \leftarrow cont + 1$
    $s \leftarrow CurrentState(cont)$
    $e \leftarrow EventDetector()$
  end while
```

8.3 Experimental Setup and Results

Activities performed in a kitchen have been studied and included in many works in the past like in [Hoey *et al.*, 2011; Patterson *et al.*, 2005] and some of them are in public datasets like [Patterson *et al.*, 2005]. However, none of the public datasets are suitable to test the system developed in this thesis, since they do not provide enough information to classify the actions that compose the activities stored in the datasets. For instance, in the case of [Patterson *et al.*, 2005], the data is labeled with the name of the activities, *Making Coffee* or *Setting the table*; but not the labels of the actions that compose those activities, *switch on the coffee maker* or *put down a plate*; which is what we need. For that reason we have designed a sensor network focused on getting that information. So, in order to test the system, a dataset has been generated with the task of making an omelette and the actions that compose this activity. The actions are *open*, *close*, *get-out*, *put-away*, *pick-up*, *put-down*, *crack-egg*, *transfer*, *switch-on*, *switch-off*, *fry*, *beat* and *null*. *null* is assigned to the events not involved in any action. A detailed description of this dataset can be found in Section 4.2.

8.3.1 ASRA. Experiment Description

The ASRA module generates sequences of actions and states. Since the states are generated directly from the sensor readings, in this section we are going to test the capability of ASRA to recognize the actions that the module will generate later. In order to test this module, we considered two different models. In the first configuration, called `SingleValues`, we generated a vector where all its positions except for the last one are binary. The positions represent the values of each sensor that appeared in the dataset. So, there is a position for each magnetic sensor and a position for each of the objects that the RFID readers can detect. Also, there is a position for every location of each object detected with the cameras (the hand of the user is not considered a location, because it is detected with RFIDs) and also a position for every different state in which an ingredient can be. Finally, there is a position for each RFID tag placed in a piece of furniture. A last element is included, indicating the index of the last position that changed. The generated vector contained 141 elements. It was computed as (10 places) + (10 places × 9 objects) + (2 RFID's × 9 objects) + (2 eggs × 4 states) + (1 salt × 2 states) + (1 oil × 2 states) + (10 RFID places) + 1 = 141 elements. But some of the elements never change. For instance, the fry-pan or the eggs are never placed in the drawers. So, those never-change values were removed and the final vector used for the classification contained 59 positions. Next, an example is shown: 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 19, open

So, the inputs of this configuration are instances composed of 59 components: the first 57 positions contain binary values; the 58th position has a value between 1 and 57 to indicate the last position that changed; and the last component contains the class, the executed action. The outputs are the actions executed at each event. For example, consider a magnetic sensor placed in a cabinet, one RFID reader and two objects (e.g., two eggs) that can be in two states (e.g., raw and cracked) in an environment with two locations to place the object (e.g., a cabinet and a working surface) and n cameras to monitor the objects. Notice that the number of cameras is irrelevant; one camera could be enough but any number of cameras could be used to detect the location and the state of the objects. Then, the vector generated for the event that was detected by a camera when the object was dropped on the working surface would be $< 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 5, putdown >$. In the example, the first element would represent the value of the magnetic sensor (cabinet closed), the second and third positions indicate the location of one egg, and the next two positions are the location of the other egg. The next three elements, the sixth, seventh and eighth, represent whether the RFID detects the object (no object is detected) or the cabinet. In the example, both eggs are in the second location (working surface) and the cabinet is not detected by the RFIDs. The ninth and tenth elements indicate the state of one egg and the next two elements indicate the state of the other egg. Both eggs are raw. Finally, the last numerical position indicates that the fifth element of the vector was the last one that changed during the event that generated the instance. So, the second egg was put-down on the kitchen top.

We try to use the same information that the planning operator would contain in order to recognize the action associated to the operator.

The second configuration, called `ObjectInvolved`, uses just the objects involved in the event that created the instance. So, whenever an event is detected, the sensors readings related to the objects involved in the event are kept and the rest are discarded, set to zero. Thus, when a specific object is picked up, a vector is generated like in the `SingleValues` configuration, but the rest of information in the vector about other objects is set to zero. For example, when a plate is put down on the kitchen top, an event is generated when the RFID stops detecting the plate. Then, the objects involved are the plate, which changes from being hold to be on the kitchen top, the kitchen top, and the RFID that changed from detecting the plate to detecting nothing. The kitchen top never changes, so none of the positions have specific information about it. So, the vector would contain a "1" in the position of the vector that indicates that the plate is on the kitchen top. When an RFID detects nothing, all the positions of the vector that contain information about that RFID are set to zero. Thus, the vector generated would contain just the "1" that indicates that the plate is on the kitchen top. Therefore, we focus the learning process on the objects currently used, eliminating the information about the rest of them. Similarly, planning operators have parameters where the objects affected by the operator are defined.

We have included an attribute to indicate the index of the position that was changed by the previous event. So, the vector is similar to the one generated for the `SingleValues` configuration but for the penultimate element, which indicates the element that was changed by the previous event. The machine learning algorithm could use the position that changed in the past event and in the current event to classify the actions. So, the inputs of the second configuration are instances composed of 62 positions: the first 59 positions contain binary values; the 58th position has a value between 1 and 57 to indicate the last position that changed; the 59th position has a value between 1 and 57 to indicate the position that changed in the previous instance; and the last position defines the class, the executed action. The outputs are the same as in the first configuration; the actions executed at each event. For example, given the previous example, the generated vector would be: $< 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 5, 7, \mathit{putdown} >$. The differences in the third and ninth positions are due to the fact that those elements do not represent information about the object involved in the current event. The last numerical position indicates that the element that changed in the previous event was the seventh.

After generating all the instances, we used the classifiers described in 7.3: PART, J48, kNN, RF, SVM, NB, *bayesian network* (BN), HNB, ALR, and DT. We have used their implementation in the Weka toolkit.

ASRA uses the algorithm EBAR presented in Chapter 7 to classify the actions with $m = 1$ since the actions that we are classifying have a short duration and many of them generate only one event.

8.3.2 ASRA. Experimental Results

In summary, we have performed experiments to check the performance of the ASRA module. The dataset generated for testing was composed of 1720 instances. Both representations generated the same number of instances since the segmentation method employed was the same in both cases. We have used two metrics to evaluate the models using 10-fold cross-validation for estimating the error: precision and recall averaged over all activities.

Then, we selected the best classifier according to the employed metric to build an AP domain. Table 8.9 shows the precision and recall of every generated model and the time in seconds that it took to classify all the instances of the dataset employing the models. The time is an important measure since it shows whether the model is fast enough classifying instances to be used in real time. The best model is marked in bold.

As it can be seen, the differences between `ObjectInvolved` and `SingleValues` are significant in most models. Results obtained by the `ObjectInvolved` configuration are

Table 8.9: Precision, recall and time performances in seconds of the learning algorithms used in ASRA.

Precision										
	PART	J48	kNN	RF	SVM	NB	BN	HNB	ALR	DT
SingleValues	94.71	95.65	72.01	93.03	95.14	65.93	68.08	90.60	94.93	94.14
ObjectInvolved	98.49	**99.09**	98.12	98.16	98.90	97.20	97.45	98.84	98.85	98.70
Recall										
	PART	J48	kNN	RF	SVM	NB	BN	HNB	ALR	DT
SingleValues	92.95	93.19	58.16	90.47	91.73	78.77	80.48	91.73	87.90	83.43
ObjectInvolved	94.33	94.59	95.07	94.40	95.33	**96.48**	92.37	93.35	94.83	92.21
Time Performances										
	PART	J48	kNN	RF	SVM	NB	BN	HNB	ALR	DT
SingleValues	0.016	<**0.001**	2.000	0.031	0.281	0.125	0.141	1.828	0.015	<**0.001**
ObjectInvolved	<**0.001**	<**0.001**	0.812	0.015	0.641	0.093	0.125	0.953	0.016	<**0.001**

better in most cases, and are more consistent since all the results are over 97.00. So, focusing the data just on the objects involved in the last event permits the system a better recognition of actions in average. The model that obtained the best result, J48, is the one we selected to generate the action sequences used to build the planning action model. In addition, the chosen model is one of the fastest models tested. It is important to consider the time performances because it can prevent the system to be used in real-time. In any case, the showed times belong to the classification of all the instances of the dataset. The classification time of a single instance is even lower. So, all generated models are fast enough to be used in the system.

8.3.3 AMLA. Experiment Description

After generating sequences of actions and states with ASRA, the AMLA module is executed to generate an AP domain using all the sequences. Then, the domain is tested to check if it is able to generate valid plans and provide assistance for users to complete a task. So, in order to test AMLA, some models were generated using different thresholds. First, a threshold of 100% was used. Then, the threshold was iteratively decreased by a 5%, until it takes the value of 0%.

A common methodology to test generated domains is to use a set of available plans selecting a subset for training and the rest of plans for testing [Wang, 1995; Yang *et al.*, 2005; Mourao *et al.*, 2009]. In [Yang *et al.*, 2005] a problem generator from the IPC is used to generate problems and the MIPS planner [Edelkamp and Helmert, 2001] is used to create the plans. Then, these plans are used for training and testing. In [Mourao *et al.*, 2009], sequences of random actions and resulting states were generated from PDDL domain descriptions using a random action generator. Problems used in [Wang, 1995] are randomly generated by the author. In order to test the domains generated by AMLA we select part of the dataset to generate the domains and the other part to generate planning problems to test the domains.

We applied a two-fold cross-validation by dividing the dataset into two parts of equal size. We selected one part to generate the planning domain and used the other part to test the domain. The testing part of the dataset was used to generate 400 problems. Each problem was generated using two non-consecutive states randomly selected from the testing part of the dataset. A hand-written planning domain was created with all the actions the system has to learn automatically for comparison purposes. This domain is shown in Annex A.

The AP problems have been executed using the MetricFF [Hoffmann, 2003] planner. The next section presents the results.

8.3.4 AMLA. Experimental Results

The best model for ASRA was selected and used to generate the sequences of actions and states that feed the AMLA module. Then, the AMLA module was used to generate AP domains. Figure 8.4 shows the percentage of solved problems by each domain generated using different thresholds. The hand-written domain was capable of solving 100% of the problems. Figure 8.5 shows the number of learned operators identical to the hand-written domain that contains 12 planning operators.

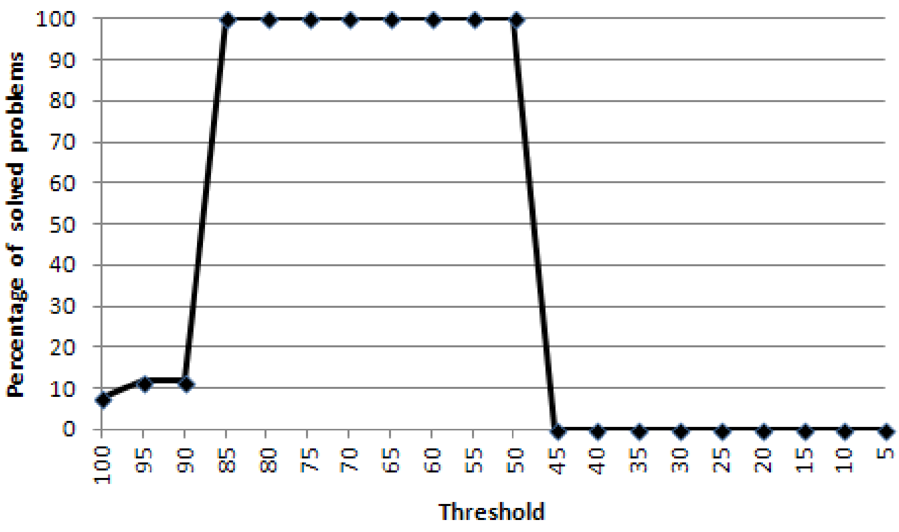

Figure 8.4: Percentage of solved problems by each domain.

As we can see in Figure 8.4, the domains generated using as threshold values from 85% to 50% are capable of solving all the problems generated. However, the number of identical operators ranges from 9 to 11. The 12 operators are never learned together.

When the threshold has a too high value, in many cases the predicates included in the preconditions let the operators be used in circumstances in which they should not. For that reason, many problems are solved even when some operators are not correct since not all preconditions that the operators need are included. For example, frying the omelette when the burner is off or beating the egg when it still has the shell.

On the other hand, when the threshold has a value too low, lower than 50%, the generated operators included too many predicates in the preconditions of some operators to be executed. For that reason, the domains generated using as a threshold values between 45% and 5% do not solve any problem.

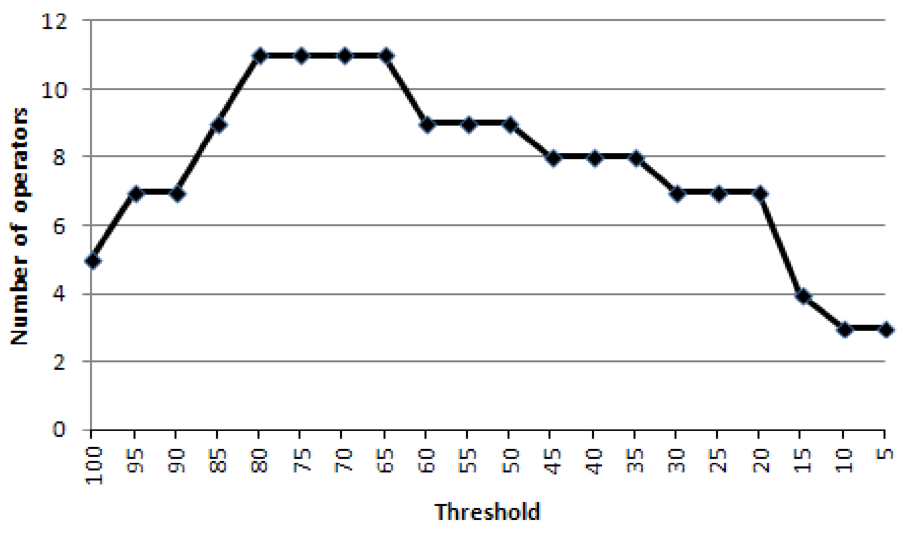

Figure 8.5: Number of learned operators identical to the hand-written domain.

The best generated domains were those with a threshold between 80% and 65% which learn all but one operator identical to the hand-written operators and solve all problems. The only action which is not learned correctly is *put-away* which is learned with less effects and less preconditions. This is due to the fact that the action *put-away* is erroneously classified as *put-down* in many cases and some of those cases are needed to learn the whole preconditions and effects. The action *put-away* is the one that the user takes when placing something in the cabinets or drawers. This action is not learned correctly because the action is very similar to *put-down*. The action *put-down* is executed when the user leaves something on the kitchen top or the cooktop. The only difference between the two operators is the place where the object is left. So, the readings of the sensors are very similar and some of the events produced during the actions are identical. Since there are many more instances of *put-down*, those identical events are always classified as performed by the *put-down* action.

Using a classifier with a slightly better recall like SVM, the action is correctly learned when the threshold is set to 50% but the other operators are correctly learned between 80% and 65% as with the other classifier. Then, not all the actions were learned correctly using the same threshold. For instance, the action *transfer* is correctly learned when the threshold has values from 65% to 85%. So, if we want our system to learn the *transfer* action, *put-away* is not correctly learned and vice versa. However, using a threshold between 80% and 65% the generated domains can guide the user through the task since the action *put-away* is not needed to complete it. This is due to the fact that it is used to replace the objects in their original places and that is not needed to cook an omelette.

The results of the learning process change as the threshold is modified like in similar approaches [Yang *et al.*, 2005; Zhuo *et al.*, 2008]. Using a different threshold for each activity solves the problem and permits us to learn the complete planning action model correctly. However, the drawback is that it requires providing more inputs for the system to work and that is what we want to reduce. Another solution to the problem is to change the domain and join the actions *put-down* and *put-away* in just one action. This can be done changing the operator in order to permit it to leave objects in any place. We have included

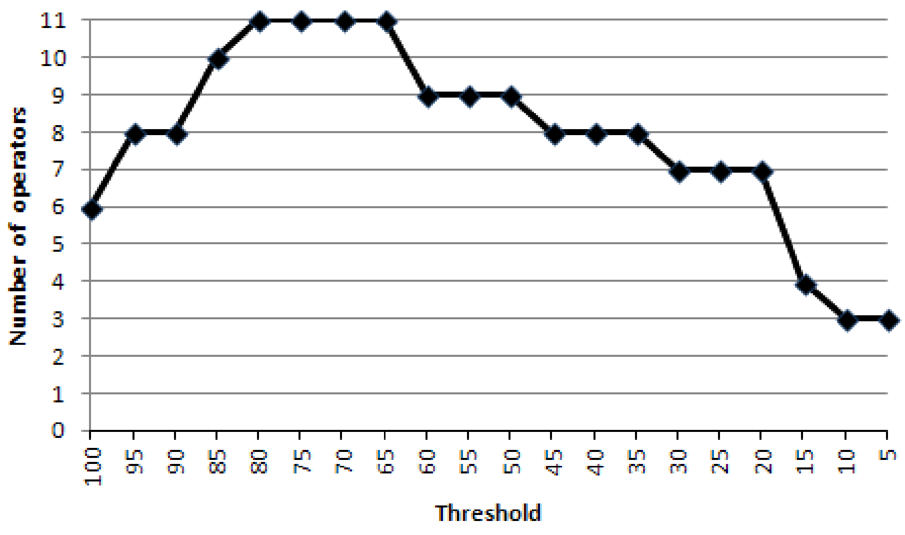

Figure 8.6: Number of learned operators identical to the hand-written domain after joining *put-down* and *put-away*.

a new configuration to test this solution in order to learn a complete domain. Figure 8.6 shows the number of learned operators identical to the hand-written domain After joining the operators *put-down* and *put-away*, it contains 11 operators.

As we can see, joining the actions permits the system to learn the entire domain when the threshold has values from 65% to 80%. Also, the generated domains can solve all the problems.

Figure 8.7 shows an example of a plan generated by a domain created by AMLA. As it is shown, the plan provides all the steps to cook an omelette. However, the plan has some minor problems. It closes the fridge at the end of the plan, leaving the door open since the beginning. Also, it leaves the door of the cabinets open until the end of the plan. This could be solved generating all the plans capable of solving the problem and using just those in which the cabinets and fridge are closed after picking up the objects from inside.

An example of learned domain can be found in Annex B. It is a domain generated with a threshold of 65% and 12 operators. The only difference with the hand-written domain is in the *put-away* operator where the precondition *(holding ?param1)* and the effect *(not (holding ?param1))* are missing.

Annex C shows the *put-down* operator after being joined to the *put-away* operator.

8.4 Recognizing and Predicting Actions Using Planning

In this section we are going to study the capability of the generated planning domains to recognize and predict the actions performed by the system's users. This was the last objective of this thesis. In order to use the planning domains for AR, we are going to check if the planning domains can recognize the actions by themselves and along with the features extracted from the sensors. The same tests are going to be executed to predict actions.

```
0: (OPEN UCUP1)
1: (OPEN FRIDGE)
2: (GETOUT BIGBOWL UCUP1)
3: (GETOUT EGG1 FRIDGE)
4: (PUTDOWN BIGBOWL KITCHENTOP)
5: (OPEN DCUP1)
6: (CRACKEGG EGG1 BIGBOWL)
7: (OPEN DCUP2)
8: (OPEN DRAWER1)
9: (PICKUP BIGBOWL KITCHENTOP)
10: (GETOUT FRYPAN DCUP1)
11: (PUTDOWN FRYPAN BURNER)
12: (GETOUT FORK1 DRAWER1)
13: (BEAT EGG1 FORK1 BIGBOWL)
14: (GETOUT OILBOTTLE DCUP2)
15: (TRANSFER OIL OILBOTTLE FRYPAN)
16: (TRANSFER EGG1 BIGBOWL FRYPAN)
17: (SWITCHON BURNER)
18: (CLOSE UCUP1)
19: (CLOSE DCUP1)
20: (CLOSE DCUP2)
21: (CLOSE DRAWER1)
22: (CLOSE FRIDGE)
23: (FRY EGG1 FRYPAN BURNER)
24: (SWITCHOFF BURNER)
```

Figure 8.7: Example of a generated plan.

8.4.1 Performing AR Using Planning Domains

In this section we want to test the capabilities of AP for recognizing actions. To that end we generate planning problems using the state before and after each event. Then, we execute the planner and check if the first action that the plan contains is the executed action that produced the event.

After generating the planning domains, three configurations have been tested in order to check if the generated domains are able to recognize actions or at least to improve the results obtained by the ASRA system.

First, we have tested the planning domain alone. So, being s_n the current state of the system, s_{n-1} the state previous to the current one and e_n the event that changed the system state from s_{n-1} to s_n, AR is the process of recognizing the action that produced the event e_n and the transition from s_{n-1} to s_n. To that end, the information of the sensors in the state s_n and before can be used. So, for each instance created by ASRA, we generate a planning problem in which s_{n-1} is used as initial state, s_n as the goal state, and n would take the number of the event that produced the state change. In Annex D an example of one of the generated problems can be found. Then, MetricFF was used to generate a plan and the first action of such plan was the action employed to recognize the user's action. This configuration was called *Total-order*.

Next, in a configuration called *Partial-order*, we employed a partial-order plan to recognize actions instead of using a total-order plan. As opposed to total-order plans, partial-order plans are partially ordered structures of actions so the total-order plans can be obtained by linear ordering of the actions respecting the partial-order. The planning problem was generated as in the previous configuration, and the planner MetricFF was used to solve it since a variation of MetricFF is capable of generating a partial-order plan. Figure 8.8 shows an example of a partial-order plan. The number before the actions indicates the order in which the actions should be executed. In this example, five actions could be executed in the first place and there is no specific order in which the five actions should be executed. Note that a partial-order plan is different from a parallel plan, where actions in the same time step could be executed in parallel. In partial-order plans, the order in which actions in the same time step are executed is irrelevant. But, they might not necessarily be executed in parallel (as in this example).

```
0: (OPEN FRIDGE )
0: (PUTDOWN BIGBOWL KITCHENTOP )
0: (OPEN DCUP1 )
0: (OPEN DCUP2 )
0: (PICKUP FORK1 KITCHENTOP )
1: (GETOUT EGG1 FRIDGE )
1: (GETOUT FRYPAN DCUP1 )
1: (GETOUT OILBOTTLE DCUP2 )
2: (CRACKEGG EGG1 BIGBOWL )
2: (PUTDOWN FRYPAN BURNER )
2: (TRANSFER OIL OILBOTTLE FRYPAN )
2: (CLOSE DCUP1 )
2: (CLOSE DCUP2 )
2: (CLOSE FRIDGE )
3: (PICKUP BIGBOWL KITCHENTOP )
3: (BEAT EGG1 FORK1 BIGBOWL )
3: (SWITCHON BURNER )
4: (TRANSFER EGG1 BIGBOWL FRYPAN )
5: (FRY EGG1 FRYPAN )
```

Figure 8.8: Example of a generated partial-order plan.

Then, the first action or group of actions of the partial-order plan was or were employed for the action's recognition. In order to prepare an omelette, the order in which many actions are executed is not important. For example, before beating an egg it does not matter whether the user first picks up the bowl where the egg is going to be beaten or the fork that is going to be used to beat the egg.

The last configuration added the first action of the plans generated by the planner to the features employed by ASRA in order to check if the action was able to improve the classification results obtained by ASRA without the help of the planner. This configuration was called *ASRA+AP*.

Table 8.10 shows the percentage of correctly classified instances obtained by each configuration in the corresponding column. A fourth column has been added showing the result obtained by ASRA for comparison. We used the ML algorithms described in Section 8.3.1, but the results shown in the columns corresponding to the third configuration and ASRA are the results obtained by the SVM algorithm which obtained the best results.

Table 8.10: Results obtained performing AR.

	Total-order	Partial-order	ASRA+AP	ASRA
Accuracy	61.24	62.37	97.73	97.73

Table 8.10 shows that the results of the first and second configurations are quite similar. This is due to the fact that the planning problems are built using the states before and after

the event. So, the differences between the two states are very small and, in most cases, the partial-order plans and the regular plans are identical. In any case, the results are far from those obtained using the standard ML algorithms shown in the third and fourth columns of the table.

In the third configuration we put together the AR method using ML and the recognition performed using AP and check if AP can improve the results obtained employing only ML. Comparing the results of the last two columns we see that the action provided by the planner added as an attribute in ASRA+AP does not improve the results obtained using only the information coming from the sensors in ASRA.

The main reason for the difference between the results obtained by the planner and ASRA is that the 20.39% of the events are produced by the action *null* which is assigned to the events not involved in any action. These events are produced by the RFIDs attached to the pieces of furniture and fridge. When the user grabs a handle to open a cabinet, the RFID of the door is detected and one event is produced. But, no action is associated to the event, since the door has not been open yet. These actions are never classified correctly by the planner, because the action *null* is not included in the domain. Even including an action *null*, it would never be used to generate a plan, since it does not help the user to accomplish the plan. Also, the planner fails in actions that take some time to execute, such as *beating*. The planner does not use it, since the effects of the action are already present in the initial state. For instance, when the cameras have detected that the egg is beaten but the user is still beating it. Another common situation in which the planner fails is when the user performs an action that is not needed to accomplish the plan. For instance, picking up an object that is not needed or that has been already used but it is picked up in order to be moved to other place.

Once the sensors detect the effects of an action the planner does not use that action in the next generated plan since the effects produced by such action are not needed. However, the user sometimes performs the same action twice. Thus, we have tried another configuration in which we have included the action that produced the previous event in the actions returned by the partial-order plan. That way, we could recognize those repetitive actions that the planner does not use. The results improved from 62.37% to 67.25%. This new configuration permitted the system to recognize consecutive repetitive actions, but it still can not recognize an action that is repeated after performing another action. The events that produced the action *null* can be removed, since the sensors that produced it are not needed by the system. Then, removing such events, we have tested another configuration to improve the performance. The results obtained are shown in Table 8.11.

Table 8.11: Results obtained performing AR without the action *null*.

	Total-order	Partial-order	ASRA+AP	ASRA
Accuracy	76.93	84.48	97.50	97.50

As we can see, if we remove the *null* events the results obtained by the planner improve a lot, because all those events were misclassified. On the other hand, the accuracy obtained using ML decreases a bit, since ML could classify correctly many of those events and without those events the total amount of well-classified events decreases.

8.4.2 Predicting Actions

In this section we want to study the capability of AP for predicting the actions of the user. AP is capable of generating a plan in order to guide the user to reach his/her goal. In this case, we want to check if the generated plans are able to match the real behavior of the users beforehand. When AR is performed, the state after the action is available. In this section we predict the next event that will take place, so the information about the state after the event will not be available. So, being s_n the current state of the system, s_{n+1} the future state to which the system will change and e_{n+1} the event that will produce the transition from s_n to s_{n+1}, this section predicts the action that will produce the event e_{n+1}.

In the previous section, we defined an experiment to recognize the last action that the user executed. The difference between both experiments is that, to predict actions that have not happened yet, we use the last known sensors state. In the previous section, we used the last two states to recognize the action between those two states. So, the event has already happened. This is the last objective pursued in this thesis. Next, the experiment performed in order to predict the user's actions using AP is described.

Four configurations have been defined in order to test the capabilities of the generated domains for predicting actions. So, for each event found by ASRA, a planning problem was created where the initial state was the state s_n, n was the number of the event that produced the transition to s_n, the goal state was to have the omelette prepared, and the action that will produce the event e_{n+1} is the one that has to be predicted. Then, a planner was used to generate a plan employing the generated problem and a planning domain generated by AMLA and the first action of such plan was the action employed to predict the next action to be executed by the user. We called this configuration *Total-order*. In Annex E an example of one of the generated problems can be found.

In the second configuration we use partial-order plans to predict the action instead of using total-order plans. The first action (or group of actions) of the partial-order plan was (were) employed for the action's prediction. We tested whether the action that will be performed is one of the actions proposed by the partial-order plan in the first place in the configuration called *Partial-order*.

In the next configuration, called *Sensors-ML*, we use the information of the current state to generate ML instances. This way, we employ the information provided by the sensors to predict the next action. Thus, for each state of the dataset we generate one instance just like in the `SingleValues` configuration described in Section 8.3.1, but without the attribute that describes the position that was changed by the event. The event has not occurred yet, so we can not use that information. In order to build the classifying instances, we employ the information of the sensors in the state before each event where the action that modifies the state will be the class.

In the last configuration called *Total-order-sensors-ML*, we added the action provided by the planner in the first configuration to the *Sensors-ML* configuration, generating instances with one attribute more than the previous configuration.

We applied a two-fold cross-validation by dividing the dataset in two parts of equal size. One part was used to train the ML algorithms and the other for testing. Table 8.12 shows the percentage of correctly classified instances obtained by each configuration in the corresponding column. For the third and fourth configurations we used the ML algorithms described in Section 8.3.1. But we only show the results obtained by the J48 algorithm which obtained the best results.

Table 8.12 shows that the accuracy obtained using a total-order plan is quite low. This

Table 8.12: Results obtained predicting the next action.

	Total-order	Partial-Order	Sensors-ML	Total-order-sensors-ML
Accuracy	19.83	45.60	84.47	84.75

was expected since, from each state, the user can achieve his/her goal following different plans. Using partial-order plans the predictions improve, but still with results below 50%.

However, in *Sensors-ML*, the ML algorithms used just the information from the sensors and obtained a better result. The last configuration shows that the action provided by the plans improves the results obtained using only sensors although the difference is very small.

Like in the previous section 8.4.1, the low results obtained by the planner are due to actions that are executed several times, actions executed by the user that are not needed to accomplish the goal and events produced by the action *null*. Including the previous action to the partial-order plans, the results improve from 45.60% to 55.92%. Removing the events produced by the action *null*, just like in the previous section, we obtained the results shown in Table 8.13.

Table 8.13: Results obtained by the planner performing AR.

	Total-order	Partial-order	Sensors-ML	Total-order-sensors-ML
Accuracy	24.91	70.24	95.87	96.08

We can see that all the results improve and the *Total-order-sensors-ML* configuration almost reaches the results obtained in the recognition task. Thus, we can conclude that, in order to predict actions, it is better to use the information provided by the sensors along with the action provided by the planner and, under some conditions, the results can be close to those obtained recognizing actions.

8.5 Discussion

In this chapter, we presented a system for building AP domains from raw sensor data. The system called ASRA-AMLA is composed of two modules that can recognize actions and states from raw sensor data (first module), and use the sequence of actions and states generated by the first module to build an AP domain (second module).

In the process of building the system, a working environment was created in a kitchen. Two persons performed a task, cooking a omelette, and a sensor network recorded the data produced by the users performing the task. Several AP domains were modeled to assist people in the task. However, using just one threshold does not permit to learn the entire domain. To model the AP domain successfully, more than one threshold has to be used. An alternative method would be to improve the sensor network in order to recognize the actions better or even code the domain in a different way joining two actions. This last alternative permits the system to correctly learn the complete domain.

The results of the learning process change as the threshold is modified and the results may change depending on the ML algorithm used. The results are also affected by the AR system, ASRA, since using SVM instead of J48, which obtained a better precision, permitted the system to learn the action *put-away* correctly.

The AP domains generated can be used, along with a planner, to find a sequence of actions that can be used to assist people in the environment where it was built. The AP domains have also been used to predict actions and recognize actions. Although the recognition rates obtained using just AP are far from the AR system, when predicting actions, the planning domain tested showed a promising result of 70.24% when the correct events are selected. Anyhow, it is still lower than the result obtained using directly the information coming from the sensors. However, adding the action provided by the planner to the sensor data may improve the predictions. The events used by the system affect the results since, removing some events that are not needed, the accuracy of ASRA slightly decreases. But the results of the planner in the recognition process and all the results predicting actions showed a considerable improvement.

The system is easily extendible. In order to include more recipes or actions, the sensorial system has to be extended to recognize the new actions and to detect the effects that the new actions would produce. Also, the new sensor readings would have to be included and how these readings are translated into predicates. That way, the new actions would be learned and included in the planning action model.

The presented system goes beyond other works in the literature on the automatic generation of planning domains since it is capable of learning the AP domains using traces of information from noisy sensors. Systems like [Yang *et al.*, 2005; Mourao *et al.*, 2009] need the right name and parameters of each operator to be learned correctly. ASRA-AMLA uses the actions provided by a classifier and the parameters are deduced from the sensor's information. This permits our system to on-line modify the generated domains. If the user decides to change how the recipe is prepared, the system can modify the domains in order to include the changes without manual intervention, in case the sensor network is prepared to detect such changes. For instance, in case the user changes the omelette and starts to prepare a fried egg, the system would modify the action *fry* where one of the preconditions forces the egg to be beaten before being fried. This is possible if the computer vision system is trained to detect the fried egg. In addition, the system ASRA-AMLA has been tested in a real environment instead of using theoretical domains like the other approaches.

Part IV

Conclusions and Future Work

Chapter 9

Conclusions

This chapter presents the conclusions extracted from this thesis. This chapter is composed of two sections. The first section summarizes the findings obtained in each chapter of the thesis. The second section describes the contributions made by this thesis to the areas involved.

9.1 Summary

Progress on wireless sensor networks and AR, made it possible to improve the quality of life of people with disabilities. Ambient assisted living (AAL) use pervasive computing, ambient intelligence or ubiquitous computing, among other technologies, to support people with special needs stay active longer, remain socially connected and to live longer periods in their preferred environment.

In the last years AP has experimented important advances and, nowadays, automated planners are capable of generating plans of hundreds of actions in a variety of domains. Nevertheless, the manual design of planning domains is time-consuming and error-prone.

In this thesis it is argued that AR can be used to automatically generate planning domains to avoid the problems entailed by the manual design of planning operators. Moreover, it is argued that AP used along with AR can help AAL overcome the challenges it faces generating sequences of actions for providing guidance and detecting the assisted person's activity.

This thesis presents ASRA-AMLA, an architecture for integrating AR, planning domains generation and AP in real-world environments to assist people in a complete system for AAL. ASRA-AMLA starts generating user models for AR with an event based approach for the segmentation of the sensors' data. Next, a sequence of actions and states is generated by the ASRA module, and it is fed to the AMLA module, which generates the planning domains. The whole system is able to monitor the user's actions, and to offer assistance in order to complete the activities. Additionally, the system can also predict the next action to be performed by the users from each state.

Chapter 5 describes, to the best of our knowledge, one of the few AR systems that use relational learning for classifying activities using simple sensors. This system employed a representation based on first-order logic to generate a relational tree to classify activities obtaining good results. That system was a first approach to learning basic actions. Then, we tried to test the relational algorithm with public datasets and we discovered that the utilized ML algorithm did not support large datasets. For that reason, in Chapter 7 we presented EBAR, a different AR algorithm, also based on the same segmentation method. EBAR employed a propositional representation. The algorithm was tested using public datasets showing, on average, better performance than other algorithms. Chapter 6 shows a comparison between the segmentation used and the common approach showing that, in some domains, our segmentation method can obtain better results, on average. Chapter 7 went deeper into the segmentation and analyzed the effects of changing the parameters of the segmentation methods over the results. In Chapter 8, the ASRA-AMLA system was described. We have shown that ASRA-AMLA is capable of inducing operators in a real-world domain from the information provided by a sensor network. The key aspect of the system was the events-based segmentation method employed, which was similar to the way planning operators work. From that segmentation, the AR algorithms were developed and the planning operators generated. Our integration of AR, AP, and learning the planning operators has been evaluated in a new dataset generated for this thesis. Experimental results showed that the ASRA-AMLA system is able to generate planning domains to guide a person through the task that is used for testing. Also, the experiments showed that the system can predict actions and even to improve the recognition rates using the generated planning domains.

One of the advantages of ASRA-AMLA is that, to the best of our knowledge, it is the first system capable of learning planning operators in a domain where all the information provided to the generator may contain errors, in the preconditions, in the effects, and even in the name of the operators since all that information is provided by an AR system. Other systems are capable of learning operators in partially observable domains or with errors in the preconditions and effects but none of them accept errors in all the inputs, especially in the name of the operators to be learned.

We have shown the power of the developed system capable of automatically building and utilizing its own operators from sensor readings. An autonomous system that recognizes user's actions, creates operators and generates plans capable of assisting the users.

9.2 Contributions

The main contribution of the thesis is the definition of a general architecture for integrating processes of activity recognition, learning planning action models and automated planning. This architecture is based on some off-the-shelf components (AI planners and ML tools) and automatically captures knowledge from the sensor readings. The architecture can learn AR user models [Ortiz *et al.*, 2008; Ortiz *et al.*, 2011] and planning action models. Additionally, the thesis work resulted in the following set of contributions:

1. A review of the state of the art in AR [Ortiz *et al.*, 2011]. Chapter 2 of the thesis describes the main works in AR. The different works are reviewed according to different criteria such as models, sensor and applications.

2. A review of the state of the art in leaning action models for deterministic and non-deterministic planning showed in Chapter 2 of this thesis.

3. A method to segment temporal series for AR based on the events produced by the sensors. [Ortiz *et al.*, 2011] describes the method where dynamic temporal windows are generated to recognize activities. Chapter 7 employs the same segmentation method and shows examples using different types of sensors. Both works show that

the segmentation method does not present some of the limitations of the commonly used sliding-window algorithm.

4. An algorithm for AR using a relational learning model. Most of the models used for AR in the literature employ statistical models although other machine learning techniques such as SVM or J48 have been utilized in the past. To the best of our knowledge, the algorithm presented in [Ortiz *et al.*, 2008], and described in Chapter 5, is one of the first algorithms for activity recognition based on first-order logic. It employed relational decision trees learned using Tilde [Blockeel and De Raedt, 1998] and obtained good results.

5. An algorithm for AR employing attribute selection and a new attribute extracted from the events, described in Chapter 7. This new feature, the classification of the events, gives information about the boundaries of the activities. Usually, AR algorithms classify pieces of activities instead of the whole activity. With this attribute, we attempted to find the boundaries of the activities in order to classify the whole activity at once. Experiments show that it improves the classification results of some machine learning techniques.

6. The analysis of different types of segmentation methods for AR. Most of AR literature focuses on obtaining better models for the recognition of activities. In this thesis, we focus on improving the algorithms using a different type of segmentation and in Chapter 7 we showed a comparison of the commonly used sliding-window method and our method based on events. The study described how changes in the size and shift of the temporal windows affect the classification results.

7. An algorithm for automatically learning planning action models using the information provided by an AR system. The system is described in Chapter 8 and it uses the AR algorithm developed in Chapter 7 to extract the actions and states from the sensor readings. Then, the actions and states are grouped into sequences that are used to create the planning domain.

8. The analysis of the use of AP for predicting and recognizing user's actions. The planning action models built in Chapter 8 were used to predict actions and also to improve the recognition results obtained by the system. This represents a new way to predict actions and to perform AR.

9. During the realization of this thesis, a sensor network was developed and used to record a new dataset. The dataset contains the data generated by some users cooking a recipe. It was generated in order to test the algorithm for generating planning action models. The dataset will be made available to the research community, so that it can be used by other researchers.

Next, we show the international conference and journal papers where part of the contents and results of this thesis have been presented:

Lecture Notes

1. **Authors:** Javier Ortiz, Angel García-Olaya y Daniel Borrajo
   Title: *A Dynamic Sliding Window for Activity Recognition*

Conference: User Modeling, Adaptation and Personalization (UMAP'11)
Booktitle: Lecture Notes in Computer Science (LNCS)
Location: Girona, Spain
Date: 2011

Contributions in Conferences

1. **Authors:** Javier Ortiz, Angel García-Olaya y Daniel Borrajo
   Title: *A Relational Learning Approach to Activity Recognition from Sensor Readings*
   Conference: IEEE Conference on Intelligent Systems (IEEE IS'08)
   Booktitle: In Proceedings of the 4th IEEE Intelligent Systems conference
   Location: Varna, Bulgaria
   Date: 2008

Journals with impact factor

1. **Authors:** Javier Ortiz, Angel García-Olaya y Daniel Borrajo
   Title: *Using Activity Recognition for Building Planning Action Models*
   Journal: International Journal of Distributed Sensor Networks. Special Issue on Intelligent Systems in Context-Based Distributed Information Fusion
   Date: 2013
   Publisher: Hindawi Publishing Corporation

Chapter 10

Future Work

1. As far as activity recognition is concerned, it is still an open problem. Consequently, the learning techniques and whole framework employed are suitable for improvements. Thus, next we enumerate some of the ideas that arose during the realization of this thesis:

   (a) To develop a more appropriate sensor network. In this work, we used a sensor network to observe the behavior of the users. The sensors used are *RFID's*, *reed switch sensors* and *cameras*. Due to technical and temporal restrictions, cameras are employed to substitute other types of sensors. For example, cameras are used to detect the state of the appliances. This could be done more efficiently by *current sensors* that are capable of monitoring whether the appliances are consuming energy or not. Since this type of sensor is binary it would save computational time. In addition, the use of accelerometers can be studied in order to improve the results. Something that probably would benefit the system is the use of depth sensors, like Kinect [Microsoft, 2009], to track the user. This sensor could avoid the user to wear any type of sensor, which is one of the major limitations of our system. However, it is important to keep the computational and economical costs of the system to a minimum and adding more sensors would increase the cost and perhaps the computational requirements to process all the data.

   (b) To extend the system for recognizing more everyday activities. The system has been tested through the task of cooking an omelette. Although the system is easily extendable for cooking more recipes and more activities, it needs the sensors to detect the effects of each action to be recognized and generated as a planning operator. To do so with some of the ADL enumerated in Section 2.1.4 such as bathing or feeding is a challenge, since, to our knowledge, there are not sensors capable of detecting the effects of those actions directly.

   (c) To make the system capable of dealing with more than one person in the same environment. All the models and experiments performed in this thesis assumed that the user of the system is a single person. For a realistic application in a real world environment, an activity recognition system should be able to deal with people visiting the user and with environments in which multiple people live. Performing multi-person activity recognition is a challenge because the users may perform different activities at the same time and in the same place, in different places or the same activity may be carried out cooperatively.

(d) To employ automated planning to fill the gaps in the action sequences. In a real environment, the ingredients are used and replaced constantly. In order to track when those ingredients are taken out, the current system needs RFID tags to produce events. Adding an RFID tag to all the new ingredients would be inconvenient. This problem could be solved with other types of sensors. Although, sensors may stop working or fail and not produce events during long periods of time. Planning could be used to fill the activities performed between events. So, every two events, a planning problem can be created and executed by a planner to find one action or a sequence of actions. That way, the sequences generated could be evaluated to check the performance of planning reproducing human behaviors.

(e) To change the system to be capable of working in real-time. The system works simulating real-time behavior, but it uses the logs saved during the dataset recording. The system could be modified in order to be tested in real-time. This way it could check if the instances' creation and evaluation and the creation of plans are fast enough to be used to offer assistance to the users of the system in a real-world environment.

(f) The use of unsupervised algorithms to recognize the activities. The machine learning algorithms employed for activity recognition require annotated data to estimate the model parameters. This is a time consuming and error prone task and one of the biggest problems in the AR community. The use of unsupervised or semi-supervised techniques would eliminate this requirement and it would make ASRA-AMLA completely automatic avoiding the problems.

2. As far as automated planning is concerned, there are some changes that could be done in order to eliminate some of the limitations to enhance and complement the system.

   (a) To generate the planning domain from decision trees or rules generated by the activity recognition system. In order to classify activities, the AR module of the system uses several machine learning algorithms; decision trees and rules are among them. These algorithms can be employed to generate the operators using the information about the sensors contained in the models. This way the preconditions and effects of each operator may be learned from the models built to classify activities.

   (b) To generate non-deterministic domains. ASRA-AMLA generates a deterministic domain to create plans reproducing the user behavior. However, a domain in which the actions would not always success or produce different effects in different situations could be more suitable to reproduce human behavior for predicting and recognizing activities.

   (c) To build a HTN to generate plans. From the deterministic planning domain generated, a HTN could be built using some algorithms such as HTN-MAKER [Hogg *et al.*, 2008]. That way a HTN planner could be employed in order to solve more complex problems and offering assistance in more complicated domains.

   (d) To learn some of the inputs of the algorithm to generate the action model. The module that builds the planning operators needs as inputs: (1) the object types and generic predicates, (2) state-action sequences automatically generated by ASRA and (3) the objects involved in each action (parameters of the actions) in the sequences. (1) and (3) are handcrafted. Learning these inputs would benefit the system by making it completely automatic.

(e) Since plans are generated to assist the user, any improvement in this sense would benefit the system. So, for example, employing different search algorithms in different situations. *Plan repairing* and *plan reuse* for domains in which planning from scratch is expensive could be used to reduce computational time.

Chapter 11

Conclusiones

This chapter is a translation of Chapter 9 into Spanish and it has been included in this document as requirement for obtaining the International Doctor Mention.

Se ha incluido este capítulo en español como requisito para la obtención de la mención de Doctor Internacional.

Este capítulo presenta las conclusiones extraídas durante la tesis. El capítulo está compuesto por dos secciones. La primera sección resume las conclusiones obtenidas en cada capítulo de la tesis. La segunda sección describe las contribuciones realizadas durante la realización de la tesis en las áreas de investigación implicadas.

11.1 Resumen

Los avances realizados en redes de sensores y reconocimiento de actividades (AR) han hecho posible la mejora en la calidad de vida de algunas personas, como por ejemplo, personas con minusvalías o personas mayores. Ambient Assisted Living (AAL) o vida asistida por el entorno utiliza la computación pervasiva y ubicua además de la inteligencia ambiental, entre otras tecnologías, para asistir a personas con necesidades especiales y que éstas puedan permanecer activas durante más tiempo, permanecer en contacto con otras personas y poder vivir periodos más largos en su propia casa.

En los últimos años la Planificación Automática (AP) ha experimentado avances importantes y, hoy en día, los planificadores automáticos son capaces de generar planes de cientos de acciones en una amplia variedad de dominios. Sin embargo, el diseño manual de dominios es una actividad que consume mucho tiempo y es propensa a errores.

Por ello, en esta tesis se propone la utilización del reconocimiento de actividades para generar automáticamente dominios de planificación y evitar los problemas derivados del diseño manual de los mismos. Además, se demuestra que la AP utilizada junto a AR puede ayudar a superar los retos a los que se enfrenta la AAL permitiendo la detección de las actividades que los usuarios de un entorno realizan y generando secuencias de acciones para proporcionar asistencia a dichos usuarios.

Esta tesis presenta el sistema ASRA-AMLA, una arquitectura que integra AR, generación de dominios de planificación y AP en entornos reales para asistir a los usuarios de un AAL. ASRA-AMLA comienza generando modelos de usuarios para reconocer sus acciones con una aproximación basada en eventos para la segmentación de los datos proporcionados por los sensores. Después, una secuencia de acciones y estados es generada por el módulo ASRA que sirve de entrada al modulo AMLA para generar dominios de planificación. Finalmente, el sistema completo es capaz de monitorizar las acciones que el usuario realiza dentro del entorno de prueba y ofrecerle asistencia para completar la actividad realizada. El sistema propuesto también es capaz de predecir la siguiente acción que el usuario realizará desde cada estado.

El capítulo 5 describe, hasta donde sabemos, uno de los pocos sistemas de reconocimiento de actividades que utiliza aprendizaje relacional para clasificar actividades utilizando sensores simples. Este sistema emplea una representación basada en lógica de primer orden para generar un árbol relacional y clasificar actividades, obteniendo buenos resultados. Este sistema fue la primera aproximación para aprender acciones básicas. Más tarde, intentamos probar el algoritmo relacional con conjuntos de datos públicos y descubrimos que el algoritmo relacional utilizado no soportaba grandes cantidades de datos. Por ello, en el capítulo 7 presentamos un nuevo algoritmo llamado EBAR basado en el mismo método de segmentación pero que utilizaba una representación proposicional. El algoritmo fue probado utilizando conjuntos de datos públicos obteniendo mejores resultados que el resto de algoritmos probados. El capítulo 6 muestra una comparativa entre el método de segmentación basado en eventos desarrollado y el método comúnmente utilizado mostrando que, en algunos dominios, el método propuesto puede obtener mejores resultados. El capítulo 7 estudia más profundamente la segmentación analizando los efectos producidos por el cambio en los valores de los parámetros utilizados por los diferentes métodos de segmentación sobre los resultados obtenidos.

El capítulo 8 describe el sistema ASRA-AMLA propuesto para la generación automática de dominios de planificación. Se ha demostrado que el sistema desarrollado es capaz de generar dominios de planificación en un dominio real utilizando información proporcionada por una red de sensores. El aspecto principal del sistema es el método de segmentación basado en eventos empleado, el cual funciona de forma similar a la manera en la que los operadores de planificación operan. Utilizando dicha segmentación, se ha desarrollado el algoritmo empleado para el reconocimiento de las acciones de los usuarios así como para la generación de los operadores de planificación. La integración de AR, AP y el aprendizaje de los dominios ha sido evaluada utilizando un nuevo conjunto de datos generados expresamente durante esta tesis para probar el sistema desarrollado. Los resultados experimentales muestran que el sistema ASRA-AMLA es capaz de generar automáticamente dominios de planificación que pueden ser utilizados para guiar a los usuarios del sistema para completar la tarea utilizada durante las pruebas. Los experimentos también muestran que el sistema es capaz de predecir acciones e incluso mejorar los resultados del reconocimiento de acciones a través de los planes generados utilizando los dominios creados por el sistema.

ASRA-AMLA es, por lo que sabemos, el primer sistema capaz de aprender operadores de planificación en un dominio en el que toda la información proporcionada al generador proviene de un sistema de reconocimiento de actividades, por lo que puede contener errores en las precondiciones, efectos e incluso en el nombre de los operadores. Otros sistemas son capaces de aprender operadores en dominios parcialmente observables o con errores en las precondiciones y efectos de los operadores pero ninguno de ellos acepta errores en todas las entradas, en especial en el nombre del operador que se aprende, que debe ser el correcto.

Hemos mostrado la potencia del sistema desarrollado capaz de construir y utilizar operadores de planificación automáticamente a partir de las lecturas proporcionadas por una red de sensores. Un sistema autónomo que reconoce las acciones realizadas por el usuario del entorno inteligente, crea operadores de planificación que modelan el comportamiento del usuario y genera planes utilizando dichos operadores para asistirle en caso de necesitarlo.

11.2 Contribuciones

La principal contribución de esta tesis es la definición de una arquitectura general para integrar procesos de reconocimiento de actividades, planificación automática y el aprendizaje de los modelos de acciones que utilizará la planificación automática. Esta arquitectura está basada en componentes ya disponibles (planificadores automáticos y herramientas de aprendizaje automático) y automáticamente captura conocimiento a partir de las lecturas de una red de sensores. La arquitectura puede aprender modelos para el AR [Ortiz *et al.*, 2008; Ortiz *et al.*, 2011] y modelos de acciones para AP. Adicionalmente, el trabajo realizado en esta tesis contribuye en el campo del reconocimiento de actividades y en el de la planificación automática proporcionando los siguientes puntos:

1. Una revisión del estado del arte en AR [Ortiz *et al.*, 2011]. El capítulo 2 de la tesis describe los principales trabajos en AR. Los diferentes trabajos son revisados según distintos criterios tales como modelos existentes, sensores utilizados y aplicaciones en las que se han utilizado.

2. Una revisión del estado del arte en aprendizaje de modelos de acciones utilizados en Planificación Determinista y Planificación con Incertidumbre mostrado en el capítulo 2.

3. Un método de segmentación de series temporales para AR basado en los eventos producidos por una red de sensores. [Ortiz *et al.*, 2011] describe el método por el cual ventanas temporales dinámicas son generadas utilizando eventos para el reconocimiento de actividades. El capítulo 7 presenta un algoritmo que emplea el mismo método de segmentación y muestra ejemplos de su utilización utilizando distintos tipos de sensores. Ambos trabajos muestran que el algoritmo de segmentación utilizado no presenta las mismas limitaciones que el algoritmo de la ventana deslizante.

4. Un algoritmo de AR utilizando un modelo de aprendizaje relacional. La mayoría de los modelos utilizados en la literatura utilizan modelos estadísticos aunque hay unos pocos modelos más que utilizan otro tipo de algoritmos tales como SVM o J48. El algoritmo presentado en [Ortiz *et al.*, 2008] y descrito en el capítulo 5 es uno de los pocos algoritmos de AR basados en lógica de primer orden. Utiliza árboles de decisión relacionales generados por el algoritmo Tilde [Blockeel and De Raedt, 1998], obteniendo buenos resultados.

5. Un algoritmo de AR que utiliza selección de atributos y un nuevo atributo extraído de los eventos utilizados, descrito en el capítulo 7 de esta Tesis. Este nuevo atributo, una clasificación de los eventos encontrados, aporta información acerca de los límites de las acciones indicando si un evento marca el final o principio de una actividad. Normalmente, los algoritmos de AR clasifican pedazos de actividades en lugar de la actividad completa. Con este atributo intentamos encontrar el momento en el que las actividades comienzan y terminan para clasificar la actividad al completo de una sola vez. Los experimentos realizados muestran que el nuevo atributo mejora los resultados de la clasificación de las actividades utilizando algunos algoritmos de aprendizaje automático.

6. El análisis de distintos tipos de segmentación para AR. La mayoría de los trabajos presentes en la literatura se centran en obtener mejores modelos para clasificar las actividades. En esta tesis nos hemos centrado en mejorar los algoritmos de AR utilizando un tipo distinto de segmentación. Así, el capítulo 7 muestra una comparación del método de la ventana deslizante y el método de la ventana dinámica propuesta en esta tesis. El estudio muestra como cambios en los parámetros de tamaño y desplazamiento de las ventanas utilizadas pueden afectar significativamente los resultados obtenidos utilizando los mismos algoritmos de aprendizaje automático.

7. Un algoritmo para aprender automáticamente dominios de AP utilizando información proporcionada por un sistema de AR. El sistema es descrito en el capítulo 8 y utiliza el algoritmo EBAR descrito en el capítulo 7 para extraer las acciones y estados de las lecturas de los sensores. Las acciones y estados extraídos son agrupadas en secuencias para aprender los dominios de AP a partir de las acciones clasificadas y las lecturas de los sensores que componen cada estado.

8. Un método de predicción y reconocimiento de acciones utilizando AP. Los modelos de acciones de AP construidos en el capítulo 8 son utilizados para predecir y reconocer acciones así como para mejorar los resultados obtenidos por el módulo de AR. Esto representa una nueva forma de predecir acciones y realizar AR.

9. Para probar el algoritmo de generación de modelos de planificación se ha tenido que desarrollar una red de sensores y grabar un conjunto de datos nuevo. El conjunto de datos generado recoge información de un usuario cocinando una receta de cocina, una tortilla francesa. Dicho conjunto de datos se ha tenido que grabar para poder probar el algoritmo de generación de dominios de planificación. Esto es debido a que no se ha encontrado en la literatura ningún conjunto de datos adecuado para probar el sistema desarrollado. El conjunto de datos será puesto a disposición de la comunidad científica para ser utilizado por otros investigadores.

Part V

Appendix

Appendix A

Hand-written Domain

This appendix shows the hand-written domain used as baseline in the tests performed to ASRA-AMLA.

```
(define (domain Kitchen)
  (:requirements :strips :typing )
  (:types
    staticobject moveable - object
    surface furniture - staticobject
    appliance worktop - surface
    ingredient utensil - moveable
    container normalutensil - utensil
    cookingcontainer normalcontainer - container
    specialcontainer - normalcontainer
    liquidcontainer - normalcontainer
    egg - ingredient)

  (:predicates
    (in ?x - staticobject ?y - moveable)
    (inside ?x - container ?y - moveable)
    (closed ?x - staticobject)
    (opened ?x - staticobject)
    (holding ?x - object)
    (on ?x - appliance)
    (off ?x - appliance)
    (raw ?x - ingredient)
    (cracked ?x - ingredient)
    (omelette ?x - ingredient)
    (beaten ?x - ingredient))

  ( :action close
    :parameters ( ?param1 - furniture)
    :precondition
    ( and
      ( opened ?param1))
    :effect
    ( and
      ( not ( opened ?param1 ) )
      ( closed ?param1)))

  ( :action putaway
    :parameters ( ?param1 - moveable
                  ?param2 - furniture)
    :precondition
    ( and
      ( opened ?param2)
      ( holding ?param1 ))
    :effect
    ( and
      ( not ( holding ?param1 ) )
      ( in ?param2 ?param1)))


  ( :action pickup
    :parameters ( ?param1 - moveable
                  ?param2 - surface)
    :precondition
    ( and
      ( in ?param2 ?param1))
    :effect
    ( and
      ( not ( in ?param2 ?param1 ) )
      ( holding ?param1)))
( :action fry
  :parameters ( ?param1 - ingredient
              ?param2 - cookingcontainer
              ?param3 - appliance)
  :precondition
  ( and
    ( inside ?param2 oil)
    ( inside ?param2 ?param1)
    ( beaten ?param1)
    ( in ?param3 ?param2)
    ( on ?param3))
  :effect
  ( and
    ( not ( beaten ?param1 ) )
    ( omelette ?param1)))


( :action transfer
  :parameters ( ?param1 - ingredient
              ?param2 - container
              ?param3 - container)
  :precondition
  ( and
    ( holding ?param2)
    ( inside ?param2 ?param1))
  :effect
  ( and
    ( not ( inside ?param2 ?param1 ) )
    ( inside ?param3 ?param1)))


( :action switchon
  :parameters ( ?param1 - appliance)
  :precondition
  ( and
    ( off ?param1)
    ( in ?param1 frypan))
  :effect
  ( and
    ( not ( off ?param1 ) )
    ( on ?param1)))


( :action switchoff
  :parameters ( ?param1 - appliance)
  :precondition
  ( and
    ( on ?param1)
    ( in ?param1 frypan))
  :effect
  ( and
    ( not ( on ?param1 ) )
    ( off ?param1)))


( :action putdown
  :parameters ( ?param1 - moveable
              ?param2 - surface)
  :precondition
  ( and
    ( holding ?param1))
  :effect
  ( and
    ( not ( holding ?param1 ) )
    ( in ?param2 ?param1)))
  ( :action open
    :parameters ( ?param1 - furniture)
    :precondition
    ( and
      ( closed ?param1))
    :effect
    ( and
      ( not ( closed ?param1 ) )
      ( opened ?param1)))


  ( :action beat
    :parameters ( ?param1 - ingredient
                  ?param2 - normalutensil
                  ?param3 - specialcontainer)
    :precondition
    ( and
      ( cracked ?param1)
      ( inside ?param3 ?param1)
      ( holding ?param2))
    :effect
    ( and
      ( not ( cracked ?param1 ) )
      ( beaten ?param1)))


  ( :action crackegg
    :parameters ( ?param1 - egg
                  ?param2 - specialcontainer)
    :precondition
    ( and
      ( raw ?param1)
      ( holding ?param1)
      ( in kitchentop ?param2))
    :effect
    ( and
      ( not ( raw ?param1 ) )
      ( not ( holding ?param1 ) )
      ( inside ?param2 ?param1)
      ( cracked ?param1)))


  ( :action getout
    :parameters ( ?param1 - moveable
                  ?param2 - furniture)
    :precondition
    ( and
      ( opened ?param2)
      ( in ?param2 ?param1))
    :effect
    ( and
      ( not ( in ?param2 ?param1 ) )
      ( holding ?param1)))

)
```

Appendix B

AMLA Generated Domain

This appendix shows a domain generated by ASRA-AMLA.

```
(define (domain Kitchen)
  (:requirements :strips :typing )
  (:types
    staticobject moveable - object
    surface furniture - staticobject
    appliance worktop - surface
    ingredient utensil - moveable
    container normalutensil - utensil
    cookingcontainer normalcontainer - container
    specialcontainer - normalcontainer
    liquidcontainer - normalcontainer
    egg - ingredient)

  (:predicates
    (in ?x - staticobject ?y - moveable)
    (inside ?x - container ?y - moveable)
    (closed ?x - staticobject)
    (opened ?x - staticobject)
    (holding ?x - object)
    (on ?x - appliance)
    (off ?x - appliance)
    (raw ?x - ingredient)
    (cracked ?x - ingredient)
    (omelette ?x - ingredient)
    (beaten ?x - ingredient))

  ( :action close
    :parameters ( ?param1 - furniture)
    :precondition
    ( and
      ( opened ?param1))
    :effect
    ( and
      ( not ( opened ?param1 ) )
      ( closed ?param1)))

  ( :action putaway
    :parameters ( ?param1 - moveable
                  ?param2 - furniture)
    :precondition
    ( and
      ( opened ?param2))
    :effect
    ( and
      ( in ?param2 ?param1)))


  ( :action pickup
    :parameters ( ?param1 - moveable
                  ?param2 - surface)
    :precondition
    ( and
      ( in ?param2 ?param1))
    :effect
    ( and
      ( not ( in ?param2 ?param1 ) )
      ( holding ?param1)))
( :action fry
  :parameters ( ?param1 - ingredient
                ?param2 - cookingcontainer
                ?param3 - appliance)
  :precondition
  ( and
    ( inside ?param2 oil)
    ( inside ?param2 ?param1)
    ( beaten ?param1)
    ( in ?param3 ?param2)
    ( on ?param3))
  :effect
  ( and
    ( not ( beaten ?param1 ) )
    ( omelette ?param1)))


( :action transfer
  :parameters ( ?param1 - ingredient
                ?param2 - container
                ?param3 - container)
  :precondition
  ( and
    ( holding ?param2)
    ( inside ?param2 ?param1))
  :effect
  ( and
    ( not ( inside ?param2 ?param1 ) )
    ( inside ?param3 ?param1)))


( :action switchon
  :parameters ( ?param1 - appliance)
  :precondition
  ( and
    ( off ?param1)
    ( in ?param1 frypan))
  :effect
  ( and
    ( not ( off ?param1 ) )
    ( on ?param1)))


( :action switchoff
  :parameters ( ?param1 - appliance)
  :precondition
  ( and
    ( on ?param1)
    ( in ?param1 frypan))
  :effect
  ( and
    ( not ( on ?param1 ) )
    ( off ?param1)))


( :action putdown
  :parameters ( ?param1 - moveable
                ?param2 - surface)
  :precondition
  ( and
    ( holding ?param1))
  :effect
  ( and
    ( not ( holding ?param1 ) )
    ( in ?param2 ?param1)))
  ( :action open
    :parameters ( ?param1 - furniture)
    :precondition
    ( and
      ( closed ?param1))
    :effect
    ( and
      ( not ( closed ?param1 ) )
      ( opened ?param1)))


  ( :action beat
    :parameters ( ?param1 - ingredient
                  ?param2 - normalutensil
                  ?param3 - specialcontainer)
    :precondition
    ( and
      ( cracked ?param1)
      ( inside ?param3 ?param1)
      ( holding ?param2))
    :effect
    ( and
      ( not ( cracked ?param1 ) )
      ( beaten ?param1)))


  ( :action crackegg
    :parameters ( ?param1 - egg
                  ?param2 - specialcontainer)
    :precondition
    ( and
      ( raw ?param1)
      ( holding ?param1)
      ( in kitchentop ?param2))
    :effect
    ( and
      ( not ( raw ?param1 ) )
      ( not ( holding ?param1 ) )
      ( inside ?param2 ?param1)
      ( cracked ?param1)))


  ( :action getout
    :parameters ( ?param1 - moveable
                  ?param2 - furniture)
    :precondition
    ( and
      ( opened ?param2)
      ( in ?param2 ?param1))
    :effect
    ( and
      ( not ( in ?param2 ?param1 ) )
      ( holding ?param1)))

)
```

Appendix C

Put-down operator after being joined to put-away

This appendix shows the *put-down* operator after being merged with the *put-away* operator.

```
( :action putdown
    :parameters ( ?param1 - moveable
                  ?param2 - staticobject)
    :precondition
    ( and
      ( holding ?param1))
    :effect
    ( and
      ( not ( holding ?param1 ) )
      ( in ?param2 ?param1)))
```

Appendix D

Generated Planning Problem for Activity Recognition

This appendix shows a generated planning problem used to recognize actions using planning.

```
(define (problem Omelette)
 (:domain Kitchen)
  (:objects
    ucup1 ucup2 dcup1 dcup2 drawer1 fridge - furniture
    burner - appliance
    kitchentop sink - worktop
    oil salt - ingredient
    egg1 - egg
    cookingcontainer normalcontainer - container
    specialcontainer - normalcontainer
    liquidcontainer - normalcontainer
    knife1 fork1 - normalutensil
    plate1 salter - normalcontainer
    oilbottle - liquidcontainer
    bigbowl - specialcontainer
    frypan - cookingcontainer
  )
  (:init
    (closed drawer1)
    (closed dcup1)
    (closed fridge)
    (closed dcup2)
    (closed ucup1)
    (closed ucup2)
    (in dcup1 frypan)
    (in kitchentop plate1)
    (in kitchentop salter)
    (in kitchentop fork1)
    (in kitchentop bigbowl)
    (in dcup2 oilbottle)
    (in fridge egg1)
    (inside salter salt)
    (inside oilbottle oil)
    (off burner)
    (raw egg1)
  )

  (:goal
    (and
      (closed drawer1)
      (closed dcup1)
      (closed fridge)
      (closed dcup2)
      (closed ucup1)
      (opened ucup2)
      (in dcup1 frypan)
      (in kitchentop plate1)
      (in kitchentop salter)
      (in kitchentop fork1)
      (in kitchentop bigbowl)
      (in dcup2 oilbottle)
      (in fridge egg1)
      (inside salter salt)
      (inside oilbottle oil)
      (off burner)
      (raw egg1)
    )
  )
)
```

Appendix E

Generated Planning Problem For Predicting Actions

This appendix shows a generated planning problem used to predict actions using planning.

```
(define (problem Omelette)
 (:domain Kitchen)
  (:objects
    ucup1 ucup2 dcup1 dcup2 drawer1 fridge - furniture
    burner - appliance
    kitchentop sink - worktop
    oil salt - ingredient
    egg1 - egg
    cookingcontainer normalcontainer - container
    specialcontainer - normalcontainer
    liquidcontainer - normalcontainer knife1 fork1 - normalutensil
    plate1 salter - normalcontainer
    oilbottle - liquidcontainer
    bigbowl - specialcontainer
    frypan - cookingcontainer
  )
  (:init
    (closed drawer1)
    (closed dcup1)
    (closed fridge)
    (closed dcup2)
    (closed ucup1)
    (closed ucup2)
    (in dcup1 frypan)
    (in kitchentop plate1)
    (in kitchentop salter)
    (in kitchentop fork1)
    (in kitchentop bigbowl)
    (in dcup2 oilbottle)
    (in fridge egg1)
    (inside salter salt)
    (inside oilbottle oil)
    (off burner)
    (raw egg1)
  )

  (:goal
    (and
      (closed ucup1)
      (closed ucup2)
      (closed dcup1)
      (closed dcup2)
      (closed drawer1)
      (closed fridge)
      (omelette egg1)
    )
  )
)
```

Appendix F

Publications

Next, we show all the international conference and journal papers that were published during the development of this thesis; some of them not related with the thesis:

Lecture Notes

1. **Authors:** Ortiz, J.; García-Olaya, A.; and Borrajo, D.
   Title: *A Dynamic Sliding Window for Activity Recognition*
   Conference: User Modeling, Adaptation and Personalization (UMAP'11)
   Booktitle: Lecture Notes in Computer Science (LNCS)
   Location: Girona, Spain
   date: 2011

Contributions in Conferences

1. **Authors:** Ortiz, J.; García-Olaya, A.; and Borrajo, D.
   Title: *A Relational Learning Approach to Activity Recognition from Sensor Readings*
   Conference: IEEE Conference on Intelligent Systems (IEEE IS'08)
   Booktitle: In Proceedings of the 4th IEEE Intelligent Systems conference
   Location: Varna, Bulgaria
   Date: 2008

2. **Authors:** Fernández, S.; Fernández, F.; Sánchez, A.; Rosa, T. de la ; Ortiz, J.; Borrajo, D.; and Manzano, D.
   Title: *On Compiling Data Mining Tasks to PDDL*
   Conference: International Conference on Automated Planning and Scheduling (ICAPS'09)
   Booktitle: In Proceedings of International Competition on Knowledge Engineering for Planning and Scheduling
   Location: Thessaloniki, Greece
   Date: 2009

3. **Authors:** Ortiz, J.; Suarez, R.; Rosa, T. de la ; Fernandez, S.; Fernandez, F.; Borrajo, D.; and Manzano, D.
   Title: *Planning for Data Mining Tool (PDM)*
   Conference: International Conference on Automated Planning and Scheduling (ICAPS'10) Demonstrations and Exhibits
   Booktitle: In Proceedings of the System Demonstrations

Location: Toronto, Canada
Date: 2010

4. **Authors:** Fernandez, S.; Suarez, R.; Rosa, T. de la ; Ortiz, J.; Fernandez, F.; Borrajo, D.; and Manzano, D.
Title: *Improving the Execution of KDD Workflows Generated by AI Planners*
Conference: European Conference on Artificial Intelligence (ECAI'10)
Booktitle: In Proceedings of the ECAI'10 3rd Planning to Learn Workshop (PlanLearn)
Location: Lisboa, Portugal)
Date: 2010

Journals with impact factor

1. **Authors:** Ortiz, J.; García-Olaya, A.; and Borrajo, D.
Title: *Using Activity Recognition for Building Planning Action Models*
Journal: International Journal of Distributed Sensor Networks. Special Issue on Intelligent Systems in Context-Based Distributed Information Fusion
Date: 2013
Publisher: Hindawi Publishing Corporation

2. **Authors:** Fernández, S.; Rosa, T. de la ; Fernández, F.; Suárez, R.; Ortiz, J.; Borrajo, D.; and Manzano, D.
Title: *Using Automated Planning for Improving Data Mining Processes*
Journal: Knowledge Engineering Review Journal
Date: 2013
Publisher: Cambridge University Press

Bibliography

[Aha *et al.*, 1991] D.W. Aha, D. Kibler, and M.K. Albert. Instance-based learning algorithms. *Machine learning*, 6(1):37–66, 1991.

[Allen, 1983] J.F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[Amft *et al.*, 2007] O. Amft, C. Lombriser, T. Stiefmeier, and G. Tröster. Recognition of user activity sequences using distributed event detection. In *EuroSSC 2007: Proceedings of the 2nd European Conference on Smart Sensing and Context*, volume 4793 of *Lecture Notes in Computer Science*, pages 126–141. Springer, October 2007.

[Amir and Chang, 2008] E. Amir and A. Chang. Learning partially observable deterministic action models. *Journal of Artificial Intelligence Research*, 33(1):349–402, 2008.

[Andrew *et al.*, 2007] A. Andrew, Y. Anokwa, K. Koscher, J. Lester, and G. Borriello. Context to make you more aware. In *Proceedings of the 27th International Conference on Distributed Computing Systems Workshops*, ICDCSW '07, page 49. IEEE Computer Society, 2007.

[Anliker *et al.*, 2004] U. Anliker, J.A. Ward, P. Lukowicz, G. Troster, F. Dolveck, M. Baer, F. Keita, E.B. Schenker, F. Catarsi, L. Coluccini, A. Belardinelli, D. Shklarski, M. Alon, E. Hirt, R. Schmid, and M. Vuskovic. AMON: a wearable multiparameter medical monitoring and alert system. *Information Technology in Biomedicine, IEEE Transactions on*, 8(4):415–427, 2004.

[Ashbrook *et al.*, 2005] D. Ashbrook, T. Westeyn, and T. Starner. Dancing in the streets: Smartphones and gaming. In *Proceedings of the Workshop on Ubiquitous Entertainment and Games at the 7 th International Conference on Ubiquitous Computing.* Citeseer, 2005.

[Aylward and Paradiso, 2006] R. Aylward and J.A. Paradiso. Sensemble: a wireless, compact, multi-user sensor system for interactive dance. In *Proceedings of the 2006 Conference on New Interfaces for Musical Expression*, pages 134–139. IRCAM-Centre Pompidou, 2006.

[Bao and Intille, 2004] L. Bao and S.S. Intille. Activity recognition from user-annotated acceleration data. *Lecture Notes in Computer Science*, pages 1–17, 2004.

[Beaudin *et al.*, 2007] J.S. Beaudin, S.S. Intille, E.M. Tapia, R. Rockinson, and M.E. Morris. Context-sensitive microlearning of foreign language vocabulary on a mobile device. In *Proceedings of the 2007 European Conference on Ambient Intelligence*, pages 55–72. Springer-Verlag, 2007.

[Begole *et al.*, 2003] J.B. Begole, J.C. Tang, and R. Hill. Rhythm modeling, visualizations and applications. In *Proceedings of the 16th annual ACM Symposium on User interface software and technology*, pages 11–20. ACM, 2003.

[Benbasat and Paradiso, 2002] A. Benbasat and J. Paradiso. An inertial measurement framework for gesture recognition and applications. *Gesture and Sign Language in Human-Computer Interaction*, pages 77–90, 2002.

[Blázquez *et al.*, 2012] G. Blázquez, A. Berlanga, and J.M. Molina. Incontexto: multisensor architecture to obtain people context from smartphones. *International Journal of Distributed Sensor Networks*, 2012.

[Blockeel and De Raedt, 1998] H. Blockeel and L. De Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, 1998.

[Blockeel *et al.*, 2006] H. Blockeel, L. Dehaspe, J. Ramon, J. Struyf, A. Van Assche, C. Vens, and D. Fierens. The ace data mining system: User's manual. http://www.cs.kuleuven.be/ dtai/ACE/, 2006.

[Bobick and Ivanov, 1998] A.F. Bobick and Y.A. Ivanov. Action recognition using probabilistic parsing. In *Computer Vision and Pattern Recognition, 1998. Proceedings. 1998 IEEE Computer Society Conference on*, pages 196–202. IEEE, 1998.

[Boger *et al.*, 2005] J. Boger, P. Poupart, J. Hoey, C. Boutilier, G. Fernie, and A. Mihailidis. A decision-theoretic approach to task assistance for persons with dementia. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, volume 19, page 1293. Citeseer, 2005.

[Bourke *et al.*, 2007] A.K. Bourke, J.V. O'Brien, and G.M. Lyons. Evaluation of a threshold-based tri-axial accelerometer fall detection algorithm. *Gait and Posture*, 26(2):194–199, 2007.

[Bradski, 2000] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[Brady *et al.*, 2005] S. Brady, L.E. Dunne, R. Tynan, D. Diamond, B. Smyth, and G.M.P. O'Hare. Garment-based monitoring of respiration rate using a foam pressure sensor. In *Proceedings of the 9th IEEE International Symposium on Wearable Computers*, pages 214–215. IEEE, 2005.

[Breiman, 2001] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[Bulling *et al.*, 2009] A. Bulling, D. Roggen, and G. Tröster. Wearable eog goggles: Seamless sensing and context-awareness in everyday environments. *Journal of Ambient Intelligence and Smart Environments*, 1(2):157–171, 4 2009.

[Chambers *et al.*, 2002] G.S. Chambers, S. Venkatesh, G.A.W. West, and H.H. Bui. Hierarchical recognition of intentional human gestures for sports video annotation. In *Proceedings of the 16th International Conference on Pattern Recognition*, volume 2, pages 1082–1085. IEEE, 2002.

[Chang *et al.*, 2007] K. Chang, M. Chen, and J. Canny. Tracking free-weight exercises. *UbiComp 2007: Ubiquitous Computing*, pages 19–37, 2007.

[Chawla, 2010] N.V. Chawla. Data mining for imbalanced datasets: An overview. *Data Mining and Knowledge Discovery Handbook*, pages 875–886, 2010.

[Chen *et al.*, 2005] J. Chen, A.H. Kam, J. Zhang, N. Liu, and L. Shue. Bathroom activity monitoring based on sound. *Pervasive Computing*, pages 47–61, 2005.

[Chmielewski *et al.*, 2011] M. Chmielewski, K. Wilkos, M. Wilkos, J. Lewandowski, and P. Stapor. Biomedical sensor analysis using mobile technologies for cardiovascular disease identification-a case study. *Man-Machine Interactions 2*, pages 127–136, 2011.

[Choi and Langley, 2005] D. Choi and P. Langley. Learning teleoreactive logic programs from problem solving. *Inductive Logic Programming*, pages 51–68, 2005.

[Choudhury and Pentland, 2003] T. Choudhury and A. Pentland. Sensing and modeling human networks using the sociometer. In *Proceedings of the 7th IEEE International Symposium on Wearable Computers*, volume 5, page 216. IEEE Computer Society, 2003.

[Choudhury *et al.*, 2006] T. Choudhury, M. Philipose, D. Wyatt, and J. Lester. Towards activity databases: Using sensors and statistical models to summarize people's lives. *IEEE Data Eng. Bull.*, 29(1):49–58, 2006.

[Choudhury *et al.*, 2008] T. Choudhury, G. Borriello, S. Consolvo, D. Haehnel, B. Harrison, B. Hemingway, J. Hightower, P. Klasnja, K. Koscher, and A. LaMarca. The mobile sensing platform: An embedded system for capturing and recognizing human activities. *IEEE Pervasive Magazine, Spec. Issue on Activity-Based Computing*, 7(2):32–41, 2008.

[Clarkson and Pentland, 1998] B. Clarkson and A. Pentland. Extracting context from environmental audio. In *Second International Symposium on Wearable Computers. Digest of Papers*, pages 154–155. IEEE, 1998.

[Clarkson and Pentland, 1999] B. Clarkson and A. Pentland. Unsupervised clustering of ambulatory audio and video. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 3037–3040. IEEE, 1999.

[Crampton *et al.*, 2007] N. Crampton, K. Fox, H. Johnston, and A. Whitehead. Dance, dance evolution: Accelerometer sensor networks as input to video games. In *IEEE International Workshop on Haptic, Audio and Visual Environments and Games (HAVE 07)*, pages 107–112. IEEE, 2007.

[Cresswell *et al.*, 2009] S. Cresswell, T.L. McCluskey, and M.M. West. Acquisition of object-centred domain models from planning examples. In *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 2009.

[Deleawe *et al.*, 2010] S. Deleawe, J. Kusznir, B. Lamb, and D.J. Cook. Predicting air quality in smart environments. *J. Ambient Intell. Smart Environ.*, 2(2):145–154, April 2010.

[Dernbach *et al.*, 2012] S. Dernbach, B. Das, N. Krishnan, B. Thomas, and D. Cook. Simple and complex activity recognition through smart phones. In *8th International Conference on Intelligent Environments (IE)*, pages 214–221. IEEE, 2012.

[Dubba *et al.*, 2010] K.S.R. Dubba, A.G. Cohn, and D.C. Hogg. Event model learning from complex videos using ilp. In *ECAI*, volume 215, pages 93–98, 2010.

[Ducatel *et al.*, 2001] K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten, and J.C. Burgelman. Scenarios for ambient Intelligence in 2010, 2001.

[Dunne *et al.*, 2006] L.E. Dunne, P. Walsh, B. Smyth, and B. Caulfield. Design and evaluation of a wearable optical sensor for monitoring seated spinal posture. In *10th IEEE International Symposium on Wearable Computers*, pages 65–68. IEEE, 2006.

[Eagle and Pentland, 2006] N. Eagle and A. Pentland. Reality mining: sensing complex social systems. *Personal and Ubiquitous Computing*, 10(4):255–268, 2006.

[Edelkamp and Helmert, 2001] S. Edelkamp and M. Helmert. Mips: The model-checking integrated planning system. *AI magazine*, 22(3):67, 2001.

[Enke, 2006] U. Enke. DanSense: Rhythmic analysis of dance movements using acceleration-onset times. *Master's thesis, RWTH Aachen University, Aachen, Germany*, 2006.

[Ermes *et al.*, 2008] M. Ermes, J. Parkka, J. Mantyjarvi, and I. Korhonen. Detection of daily activities and sports with wearable sensors in controlled and uncontrolled conditions. *IEEE Transactions on Information Technology in Biomedicine*, 12(1):20–26, 2008.

[Fikes and Nilsson, 1971] R.E. Fikes and N.J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2:189–208, 1971.

[Finkenzeller, 2010] K. Finkenzeller. *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards, Radio Frequency Identification and Near-Field Communication.* Wiley, 2010.

[Fishkin *et al.*, 2004] K.P. Fishkin, B. Jiang, M. Philipose, and S. Roy. I sense a disturbance in the force: Unobtrusive detection of interactions with rfid-tagged objects. *Lecture Notes in Computer Science*, 3205:268–282, 2004.

[Fishkin *et al.*, 2005] K.P. Fishkin, M. Philipose, and A. Rea. Hands-on RFID: Wireless wearables for detecting use of objects. *Proceedings of the IEEE International Symposium on Wearable Computers (ISWC 05)*, pages 38–43, 2005.

[Fogarty *et al.*, 2006] J. Fogarty, C. Au, and S.E. Hudson. Sensing from the basement: a feasibility study of unobtrusive and low-cost home activity recognition. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*, page 100. ACM, 2006.

[Fox and Long, 2003] M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, pages 61–124, 2003.

[Fox *et al.*, 2006] M. Fox, M. Ghallab, G. Infantes, and D. Long. Robot introspection through learned hidden markov models. *Artificial Intelligence*, 170(2):59–113, 2006.

[Frank and Witten, 1998] E. Frank and I.H. Witten. Generating accurate rule sets without global optimization. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 144–151. Citeseer, 1998.

[Friedman *et al.*, 2000] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 95(2):337–407, 2000.

[García-Martínez and Borrajo, 1997] R. García-Martínez and D. Borrajo. Planning, learning, and executing in autonomous systems. *Recent Advances in AI Planning*, pages 208–220, 1997.

[Gavrila, 1999] D.M. Gavrila. The visual analysis of human movement: A survey. *Computer Vision and Image Understanding*, 73(1):82–98, 1999.

[Gerasimov, 2003] V. Gerasimov. *Every sign of life.* PhD thesis, Massachusetts Institute of Technology, 2003.

[Ghallab *et al.*, 2004] M. Ghallab, D.S. Nau, and P. Traverso. *Automated Planning: theory and practice.* Morgan Kaufmann Publishers, 2004.

[Golding and Lesh, 1999] A.R. Golding and N. Lesh. Indoor navigation using a diverse set of cheap, wearable sensors. In *The Third International Symposium on Wearable Computers, 1999. Digest of Papers*, pages 29–36. IEEE, 1999.

[Goldman *et al.*, 1999] R.P. Goldman, C.W. Geib, and C.A. Miller. A new model of plan recognition. In *Proceedings of the 1999 Conference on Uncertainty in Artificial Intelligence*, volume 13, page 14. Citeseer, 1999.

[Gómez-Romero *et al.*, 2012] J. Gómez-Romero, M.A. Serrano, M.A. Patricio, J. García, and J.M. Molina. Context-based scene recognition from visual data in smart homes: an information fusion approach. *Personal and Ubiquitous Computing*, 16(7):835–857, 2012.

[Gu *et al.*, 2009] T. Gu, Z. Wu, X. Tao, H.K. Pung, and J. Lu. epSICAR: An Emerging Patterns based approach to sequential, interleaved and Concurrent Activity Recognition. In *Proceedings of the 2009 IEEE International Conference on Pervasive Computing and Communications*, pages 1–9. IEEE Computer Society, 2009.

[Guan *et al.*, 2007] D. Guan, W. Yuan, Y. Lee, A. Gavrilov, and S. Lee. Activity recognition based on semi-supervised learning. In *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 07)*, pages 469–475. IEEE, 2007.

[Hall and Smith, 1998] M.A. Hall and L.A. Smith. Practical feature subset selection for machine learning. *Computer Science*, 98:4–6, 1998.

[Hamid *et al.*, 2005] R. Hamid, S. Maddi, A. Johnson, A. Bobick, I. Essa, and C. Isbell. Unsupervised activity discovery and characterization from event-streams. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI05).* Citeseer, 2005.

[Haya *et al.*, 2004] P.A. Haya, G. Montoro, and X. Alamán. A prototype of a context-based architecture for intelligent home environments. In *On the Move to Meaningful Internet Systems*, pages 477–491. Springer, 2004.

[Heideman *et al.*, 1985] M.T. Heideman, D.H. Johnson, and C.S. Burrus. Gauss and the history of the fast Fourier transform. *Archive for History of Exact Sciences*, 34(3):265–277, 1985.

[Heinz *et al.*, 2007] E.A. Heinz, K.S. Kunze, M. Gruber, D. Bannach, and P. Lukowicz. Using wearable sensors for real-time recognition tasks in games of martial arts-an initial experiment. In *IEEE Symposium on Computational Intelligence and Games*, pages 98–102. IEEE, 2007.

[Hoey *et al.*, 2011] J. Hoey, T. Plötz, D. Jackson, A. Monk, C. Pham, and P. Olivier. Rapid specification and automated generation of prompting systems to assist people with dementia. *Pervasive and Mobile Computing*, 7(3):299–318, 2011.

[Hoffmann, 2003] J. Hoffmann. The metric-ff planning system: translating "ignoring delete lists" to numeric state variables. *J. Artif. Int. Res.(JAIR)*, 20(1):291–341, December 2003.

[Hogg *et al.*, 2008] C. Hogg, H. Munoz-Avila, and U. Kuter. HTN-MAKER: Learning HTNs with minimal additional knowledge engineering required. In *Proceedings of AAAI*, volume 8, 2008.

[Holmes and Isbell Jr, 2005] M.P. Holmes and C.L. Isbell Jr. Schema learning: Experience-based construction of predictive action models. *Advances in Neural Information Processing Systems*, 17:585–592, 2005.

[Hong and Nugent, 2010] X. Hong and C.D. Nugent. Partitioning time series sensor data for activity recognition. In *9th International Conference on Information Technology and Applications in Biomedicine (ITAB 2009)*, pages 1–4. IEEE, 2010.

[Huynh and Schiele, 2005] T. Huynh and B. Schiele. Analyzing features for activity recognition. In *Proceedings of the 2005 joint Conference on Smart objects and ambient Intelligence: innovative context-aware services: usages and technologies*, page 163. ACM, 2005.

[Huynh and Schiele, 2006] T. Huynh and B. Schiele. Unsupervised discovery of structure in activity data using multiple eigenspaces. In *Proceedings of the Second International Conference on Location-and context-awareness*, LoCA'06, page 151. Springer-Verlag New York Inc, 2006.

[Huynh *et al.*, 2007] T. Huynh, U. Blanke, and B. Schiele. Scalable recognition of daily activities with wearable sensors. In *Proceedings of the 3rd International Conference on Location-and context-awareness*, pages 50–67. Springer-Verlag, 2007.

[Huynh *et al.*, 2008] T. Huynh, M. Fritz, and B. Schiele. Discovery of activity patterns using topic models. In *Proceedings of the 10th International Conference on Ubiquitous computing*, UbiComp '08, pages 10–19, New York, USA, 2008. ACM.

[Intille *et al.*, 2006] S. Intille, K. Larson, E. Tapia, J. Beaudin, P. Kaushik, J. Nawyn, and R. Rockinson. Using a live-in laboratory for ubiquitous computing research. *Pervasive Computing*, pages 349–365, 2006.

[Jafari *et al.*, 2007] R. Jafari, W. Li, R. Bajcsy, S. Glaser, and S. Sastry. Physical activity monitoring for assisted living at home. In *4th International Workshop on Wearable and Implantable Body Sensor Networks (BSN 2007)*, pages 213–219. Springer, 2007.

[Jiménez *et al.*, 2013] S. Jiménez, F. Fernández, and D. Borrajo. Integrating planning, execution and learning to improve plan execution. *Computational Intelligence Journal*, 29(1):1–36, 2013.

[John and Langley, 1995] G.H. John and P. Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*, pages 338–345. Morgan Kaufmann Publishers Inc., 1995.

[Junejo *et al.*, 2011] I.N. Junejo, E. Dexter, I. Laptev, and P. Pérez. View-independent action recognition from temporal self-similarities. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):172–185, 2011.

[Junker *et al.*, 2005] H. Junker, P. Lukowicz, and G. Tröster. Padnet: wearable physical activity detection network. In *Proceedings of the Seventh IEEE International Symposium on Wearable Computers (ISWC'03)*, volume 1530, pages 17–00. Citeseer, 2005.

[Junker *et al.*, 2008] H. Junker, O. Amft, P. Lukowicz, and G. Tröster. Gesture spotting with body-worn inertial sensors to detect user activities. *Pattern Recognition*, 41(6):2010–2024, 2008.

[Kahn *et al.*, 1999] J.M. Kahn, R.H. Katz, and K.S.J. Pister. Mobile networking for smart dust. In *ACM/IEEE Intl. Conf. on Mobile Computing and Networking (MobiCom 99), Seattle, WA*, pages 271–278, 1999.

[Kasten and Langheinrich, 2001] O. Kasten and M. Langheinrich. First experiences with bluetooth in the smart-its distributed sensor network. In *Workshop on Ubiquitous Computing and Communications, PACT*, volume 1, 2001.

[Kasteren *et al.*, 2008] T.V. Kasteren, N. Athanasios, G. Englebienne, and B. Kröse. Accurate activity recognition in a home setting. In *Proceedings of the 10th International Conference on Ubiquitous Computing*, pages 1–9, New York, NY, USA, 2008. ACM.

[Kasteren *et al.*, 2010a] T.V. Kasteren, G. Englebienne, and B. Kröse. An activity monitoring system for elderly care using generative and discriminative models. *Personal and Ubiquitous Computing*, 14(6):489–498, 2010.

[Kasteren *et al.*, 2010b] T.V. Kasteren, G. Englebienne, and B. Kröse. Activity recognition using semi-markov models on real world smart home datasets. *J. Ambient Intell. Smart Environ.*, 2(3):311–325, August 2010.

[Kasteren *et al.*, 2010c] T.V. Kasteren, G. Englebienne, and B. Kröse. Towards a Consistent Methodology for Evaluating Activity Recognition Model Performance. In *Pervasive Computing 2010 Workshop on How To Do Good Research In Activity Recognition*, 2010.

[Kasteren *et al.*, 2011] T.V. Kasteren, G. Englebienne, and B. Kröse. Hierarchical activity recognition using automatically clustered actions. In *Ambient Intelligence*, pages 82–91. Springer, 2011.

[Katz *et al.*, 1963] S. Katz, A.B. Ford, R.W. Moskowitz, B.A. Jackson, and M.W. Jaffe. Studies of illness in the aged. The index of ADL: a standardized measure of biological and psychosocial function. *Jama*, 185(12):914, 1963.

[Kautz, 1991] H. Kautz. A formal theory of plan recognition and its implementation. *Reasoning about plans*, 125, 1991.

[Kern *et al.*, 2003a] N. Kern, B. Schiele, H. Junker, P. Lukowicz, and G. Tröster. Wearable sensing to annotate meeting recordings. *Personal and Ubiquitous Computing*, 7(5):263–274, 2003.

[Kern *et al.*, 2003b] N. Kern, B. Schiele, and A. Schmidt. Multi-sensor activity context detection for wearable computing. *Ambient Intelligence*, pages 220–232, 2003.

[Kern *et al.*, 2004] N. Kern, S. Antifakos, B. Schiele, and A. Schwaninger. A model for human interruptability: experimental evaluation and automatic estimation from wearable sensors. In *Wearable Computers, 2004. ISWC 2004. Eighth International Symposium on*, volume 1, pages 158–165. IEEE, 2004.

[Kern *et al.*, 2007] N. Kern, B. Schiele, and A. Schmidt. Recognizing context for annotating a live life recording. *Personal and Ubiquitous Computing*, 11(4):251–263, 2007.

[Kerr *et al.*, 2011] W. Kerr, A. Tran, and P. Cohen. Activity recognition with finite state machines. In *International Joint Conference on Artificial Intelligence*, 2011.

[Kohavi and John, 1997] R. Kohavi and G.H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1):273–324, 1997.

[Kohavi, 1995] R. Kohavi. The power of decision tables. In *Proceedings of the European Conference on Machine Learning*, pages 174–189. Springer Verlag, 1995.

[Koskimaki *et al.*, 2008] H. Koskimaki, I. Juutilainen, P. Laurinen, and J. Roning. Two-level clustering approach to training data instance selection: A case study for the steel industry. In *IEEE International Joint Conference on Neural Networks*, pages 3044–3049. IEEE, 2008.

[Krause *et al.*, 2003] A. Krause, D.P. Siewiorek, A. Smailagic, and J. Farringdon. Unsupervised, dynamic identification of physiological and activity context in wearable computing. In *Proceedings of the 7th IEEE International Symposium on Wearable Computers*, page 88. Published by the IEEE Computer Society, 2003.

[Kunze *et al.*, 2006] K. Kunze, M. Barry, E.A. Heinz, P. Lukowicz, D. Majoe, and J. Gutknecht. Towards Recognizing Tai Chi–An Initial Experiment Using Wearable Sensors. *Proc. IFAWC*, 2006.

[Kwapisz *et al.*, 2011] J.R. Kwapisz, G.M. Weiss, and S.A. Moore. Activity recognition using cell phone accelerometers. *ACM SIGKDD Explorations Newsletter*, 12(2):74–82, 2011.

[Lanchas *et al.*, 2007] J. Lanchas, S. Jiménez, F. Fernández, and D. Borrajo. Learning action durations from executions. In *In Proceedings of the ICAPS'07 Workshop on Planning and Learning*. Providence, Rhode Island (USA), 2007.

[Landwehr *et al.*, 2008] N. Landwehr, B. Gutmann, I. Thon, L. De Raedt, and M. Philipose. Relational transformation-based tagging for activity recognition. *Fundamenta Informaticae*, 89(1):111–129, 2008.

[Lawton and Brody, 1969] M.P. Lawton and E.M. Brody. Assessment of older people: self-maintaining and instrumental activities of daily living. *The gerontologist*, 1969.

[Lester *et al.*, 2006] J. Lester, T. Choudhury, and G. Borriello. A practical approach to recognizing physical activities. *Pervasive Computing*, pages 1–16, 2006.

[Liao *et al.*, 2005] L. Liao, D. Fox, and H. Kautz. Location-based activity recognition using relational Markov networks. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, pages 773–778, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.

[Liao *et al.*, 2007] L. Liao, D. Fox, and H. Kautz. Extracting places and activities from gps traces using hierarchical conditional random fields. *The International Journal of Robotics Research*, 26(1):119–134, 2007.

[Lindner *et al.*, 2005] M. Lindner, M. Kalech, and G.A. Kaminka. Detecting Coordination Failures by Observing Groups: A Formal Approach. In *Proceedings of the IJCAI Workshop on Modeling Others from Observations (MOO-05).* Citeseer, 2005.

[Linz *et al.*, 2006] T. Linz, C. Kallmayer, R. Aschenbrenner, and H. Reichl. Fully untegrated EKG shirt based on embroidered electrical interconnections with conductive yarn and miniaturized flexible electronics. In *Wearable and Implantable Body Sensor Networks, 2006. BSN 2006. International Workshop on*, pages 4–26. IEEE, 2006.

[Liszka *et al.*, 2004] K.J. Liszka, M.A. Mackin, M.J. Lichter, D.W. York, D. Pillai, and D.S. Rosenbaum. Keeping a Beat on the Heart. *IEEE Pervasive Computing*, pages 42–49, 2004.

[Logan *et al.*, 2007] B. Logan, J. Healey, M. Philipose, E.M. Tapia, and S. Intille. A long-term evaluation of sensing modalities for activity recognition. *Lecture Notes in Computer Science*, 4717:483–500, 2007.

[Lombriser *et al.*, 2007] C. Lombriser, N.B. Bharatula, D. Roggen, and G. Tröster. On-body activity recognition in a dynamic sensor network. In *Proceedings of the 2nd International Conference on Body Area Networks*, pages 1–6. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007.

[Lukowicz *et al.*, 2004] P. Lukowicz, J.A. Ward, H. Junker, M. Stäger, G. Tröster, A. Atrash, and T. Starner. Recognizing Workshop activity using body worn microphones and accelerometers. *Pervasive Computing*, pages 18–32, 2004.

[Lukowicz *et al.*, 2007] P. Lukowicz, A. Timm-Giel, M. Lawo, and O. Herzog. Wearit@ work: Toward real-world industrial wearable computing. *IEEE Pervasive Computing*, pages 8–13, 2007.

[Maitland *et al.*, 2006] J. Maitland, S. Sherwood, L. Barkhuus, I. Anderson, M. Hall, B. Brown, M. Chalmers, and H. Muller. Increasing the awareness of daily activity levels with pervasive computing. In *Pervasive Health Conference and Workshops, 2006*, pages 1–9. IEEE, 2006.

[McCluskey *et al.*, 2002] T.L. McCluskey, N.E. Richardson, and R.M. Simpson. An interactive method for inducing operator descriptions. In *The 6th International Conference on Artificial Intelligence Planning Systems.* AAAI Press, 2002.

[McCluskey *et al.*, 2007] T.L. McCluskey, S.N. Cresswell, N.E. Richardson, and M.M. West. Opmaker2: efficient action schema acquisition. In *The 26th Workshop of the UK PLANNING AND SCHEDULING Special Interest Group - PlanSIG 2007*, December 2007.

[McCluskey *et al.*, 2009] T.L. McCluskey, S.N. Cresswell, N.E. Richardson, and M.M. West. Automated acquisition of action knowledge. In *International Conference on Agents and Artificial Intelligence (ICAART)*, January 2009.

[Mcdermott, 2000] D. Mcdermott. The 1998 ai planning systems competition. *AI Magazine*, 21:35–55, 2000.

[Mckeever *et al.*, 2010] S. Mckeever, J. Ye, L. Coyle, C. Bleakley, and S. Dobson. Activity recognition using temporal evidence theory. *J. Ambient Intell. Smart Environ.*, 2:253–269, August 2010.

[Medynskiy *et al.*, 2007] Y. Medynskiy, S. Gov, A. Mazalek, and D. Minnen. Wearable RFID for Play. In *Intelligent User Interfaces 2007 Tangible Play Workshop*, 2007.

[Microsoft, 2009] Microsoft. Microsoft corp. the kinect. http://www.microsoft.com/en-us/kinectforwindows/, 2009.

[Mihailidis *et al.*, 2001] A. Mihailidis, G.R. Fernie, and J.C. Barbenel. The use of Artificial Intelligence in the design of an intelligent cognitive orthosis for people with dementia. *Assistive technology: the official journal of RESNA*, 13(1):23, 2001.

[Minnen *et al.*, 2006] D. Minnen, T. Starner, M. Essa, and C. Isbell. Discovering characteristic actions from on-body sensor data. In *10th IEEE International Symposium on Wearable Computers*, pages 11–18. IEEE, 2006.

[Minnen *et al.*, 2007] D. Minnen, T. Westeyn, D. Ashbrook, P. Presti, and T. Starner. Recognizing soldier activities in the field. In *4th International Workshop on Wearable and Implantable Body Sensor Networks (BSN 2007)*, pages 236–241. Springer, 2007.

[Modayil *et al.*, 2008] J. Modayil, T. Bai, and H. Kautz. Improving the recognition of interleaved activities. In *Proceedings of the 10th International Conference on Ubiquitous computing*, pages 40–43. ACM, 2008.

[Montoye *et al.*, 1983] H.J. Montoye, R. Washburn, S. Servais, A. Ertl, J.G. Webster, and F.J. Nagle. Estimation of energy expenditure by a portable accelerometer. *Medicine & Science in Sports & Exercise*, 15(5):403, 1983.

[Mourao *et al.*, 2008] K. Mourao, R.P.A. Petrick, and M. Steedman. Using kernel perceptrons to learn action effects for planning. In *International Conference on Cognitive Systems (CogSys 2008)*, pages 45–50, 2008.

[Mourao *et al.*, 2009] K. Mourao, R.P.A. Petrick, and M. Steedman. Learning action effects in partially observable domains. In *Proceedings of the ICAPS 2009 Workshop on Planning and Learning*, pages 15–22, 2009.

[Mynatt *et al.*, 2000] E.D. Mynatt, I. Essa, and W. Rogers. Increasing the opportunities for aging in place. In *Proceedings on the 2000 Conference on Universal Usability*, pages 65–71. ACM, 2000.

[Nejati *et al.*, 2006] N. Nejati, P. Langley, and T. Konik. Learning hierarchical task networks by observation. In *Proceedings of the 23rd International Conference on Machine learning*, pages 665–672. ACM, 2006.

[Ni *et al.*, 2004] L.M. Ni, Y. Liu, Y.C. Lau, and A.P. Patil. LANDMARC: indoor location sensing using active RFID. *Wireless Networks*, 10(6):701–710, 2004.

[Nintendo, 2006] Nintendo. Nintendo wii. http://www.nintendo.com/wii, 2006.

[Niu *et al.*, 2004] W. Niu, J. Long, D. Han, and Y.F. Wang. Human activity detection and recognition for video surveillance. In *Proceedings of the IEEE Multimedia and Expo Conference*, pages 719–722. IEEE, 2004.

[Oliver and Flores-Mangas, 2006] N. Oliver and F. Flores-Mangas. Healthgear: A real-time wearable system for monitoring and analyzing physiological signals. In *International Workshop on Wearable and Implantable Body Sensor Networks*, pages 4–64. IEEE, 2006.

[Oliver *et al.*, 2002] N. Oliver, E. Horvitz, and A. Garg. Layered representations for human activity recognition. In *Proceedings of the Fourth IEEE International Conference on Multimodal Interfaces*, pages 3–8. IEEE Computer Society, 2002.

[Ortiz *et al.*, 2008] J. Ortiz, A. García-Olaya, and D. Borrajo. A Relational Learning Approach to Activity Recognition from Sensor Readings. In *4th International IEEE Conference on Intelligent Systems*, volume 3 of *IS'08.*, 2008.

[Ortiz *et al.*, 2011] J. Ortiz, A. García-Olaya, and D. Borrajo. A dynamic sliding window approach for activity recognition. In *User Modeling, Adaptation and Personalization (UMAP)*, volume 6787, pages 219–230. Springer, 2011.

[Paradiso *et al.*, 2005] R. Paradiso, G. Loriga, and N. Taccini. A wearable health care system based on knitted integrated sensors. *IEEE Transactions on Information Technology in Biomedicine*, 9(3):337–344, 2005.

[Pasula *et al.*, 2007] H.M. Pasula, L.S. Zettlemoyer, and L.P. Kaelbling. Learning symbolic models of stochastic domains. *Journal of Artificial Inteligence Research*, 29, 2007.

[Patterson *et al.*, 2004a] D.J. Patterson, D. Fox, H. Kautz, and M. Philipose. Sporadic state estimation for general activity inference. Technical report, Technical report, Intel Research Seattle and the University of Washington, 2004.

[Patterson *et al.*, 2004b] D.J. Patterson, L. Liao, K. Gajos, M. Collier, N. Livic, K. Olson, S. Wang, D. Fox, and H. Kautz. Opportunity knocks: A system to provide cognitive assistance with transportation services. *UbiComp 2004: Ubiquitous Computing*, pages 433–450, 2004.

[Patterson *et al.*, 2005] D. Patterson, D. Fox, H. Kautz, and M. Philipose. Fine-grained activity recognition by aggregating abstract object usage. In *Ninth IEEE International Symposium on Wearable Computers*, pages 44–51, 2005.

[Pentland, 1996] A.P. Pentland. Smart rooms. *Scientific American*, 274(4):54–62, 1996.

[Pham and Olivier, 2009] C. Pham and P. Olivier. Slice&dice: Recognizing food preparation activities using embedded accelerometers. *Ambient Intelligence*, pages 34–43, 2009.

[Philipose *et al.*, 2004] M. Philipose, K.P. Fishkin, M. Perkowitz, D.J. Patterson, D. Fox, H. Kautz, and D. Hahnel. Inferring activities from interactions with objects. *IEEE Pervasive Computing*, pages 50–57, 2004.

[Platt, 1999] J. Platt. Fast training of support vector machines using sequential minimal optimization. In *Fast Training of Support Vector Machines using Sequential Minimal Optimization. Advances in Kernel Methods - Support Vector Learning*, chapter Advances in kernel methods, pages 185–208. MIT Press, Cambridge, MA, USA, 1999.

[Poole, 1993] D. Poole. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64(1):81–129, 1993.

[Pynadath and Wellman, 2000] D.V. Pynadath and M.P. Wellman. Probabilistic state-dependent grammars for plan recognition. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 507–514. Morgan Kaufmann Publishers Inc., 2000.

[Pynadath, 1999] D.V. Pynadath. *Probabilistic grammars for plan recognition.* PhD thesis, The University of Michigan, 1999.

[Quinlan, 1993] J.R. Quinlan. *C4.5: programs for machine learning.* Morgan Kaufmann, 1993.

[Rabiner and Juang, 1993] L. Rabiner and B.H. Juang. *Fundamentals of speech recognition.* Prentice hall Englewood Cliffs, New Jersey, 1993.

[Rabiner, 1989] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77, pages 257–286. IEEE, 1989.

[Ramírez and Geffner, 2009] M. Ramírez and H. Geffner. Plan recognition as planning. *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 1778–1783, 2009.

[Randell and Muller, 2000] C. Randell and H. Muller. Context awareness by analysing accelerometer data. In *The Fourth International Symposium on Wearable Computers*, pages 175–176. IEEE, 2000.

[Ravi *et al.*, 2005] N. Ravi, N. Dandekar, P. Mysore, and M.L. Littman. Activity recognition from accelerometer data. In *Proceedings of the 17th Conference on Innovative Applications on Artificial Intelligence*, volume 20 of *IAAI'05*, pages 1541–1546. AAAI Press, 2005.

[Reddy and Tadepalli, 1997] C. Reddy and P. Tadepalli. Learning goal-decomposition rules using exercises. In *Proceedings of the National Conference on Artificial Intelligence*, pages 843–843. AAAI, 1997.

[Rhodes, 1997] B.J. Rhodes. The wearable remembrance agent: A system for augmented memory. *Personal and Ubiquitous Computing*, 1(4):218–224, 1997.

[Ristic *et al.*, 2004] B. Ristic, S. Arulampalam, and N. Gordon. *Beyond the Kalman filter: Particle filters for tracking applications.* Artech House Publishers, 2004.

[Ruby and Kibler, 1991] D. Ruby and D. Kibler. SteppingStone: An empirical and analytical evaluation. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 527–532, 1991.

[Ryoo and Aggarwal, 2006] M.S. Ryoo and J.K. Aggarwal. Recognition of composite human activities through context-free grammar based representation. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1709–1718, 2006.

[Schiele *et al.*, 1999] B. Schiele, N. Oliver, T. Jebara, and A. Pentland. An interactive computer vision system dypers: Dynamic personal enhanced reality system. *Computer Vision Systems*, pages 51–65, 1999.

[Shahaf and Amir, 2006] D. Shahaf and E. Amir. Learning partially observable action schemas. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 913. AAAI Press, 2006.

[Shannon, 1948] C. E. Shannon. A mathematical theory of communication. *Bell Sys. Tech. J.*, 27:379–423, 623–656, 1948.

[Singer and Warmuth, 1999] Y. Singer and M.K. Warmuth. Batch and on-line parameter estimation of Gaussian mixtures based on the joint entropy. *Advances in Neural Information Processing Systems*, pages 578–584, 1999.

[Sony, 2009] Sony. Sony corp. playstation@move. http://playstationmove.com/index.html, 2009.

[Starner *et al.*, 1997] T. Starner, J. Weaver, and A. Pentland. A wearable computer-based American sign Language Recogniser. *Personal and Ubiquitous Computing*, 1(4):241–250, 1997.

[Starner *et al.*, 1999] T. Starner, B. Rhodes, J. Weaver, and A. Pentland. Everyday-use Wearable Computers. In *International Symposium on Wearable Computers*. Citeseer, 1999.

[Stiefmeier *et al.*, 2006] T. Stiefmeier, G. Ogris, H. Junker, P. Lukowicz, and G. Tröster. Combining motion sensors and ultrasonic hands tracking for continuous activity recognition in a maintenance scenario. In *10th IEEE International Symposium on Wearable Computers*, pages 97–104. IEEE, 2006.

[Stiefmeier *et al.*, 2007] T. Stiefmeier, D. Roggen, and G. Tröster. Gestures are strings: Efficient online gesture spotting and classification using string matching. In *Proceedings of the ICST 2nd International Conference on Body Area Networks*, pages 1–8. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007.

[Stiefmeier *et al.*, 2008] T. Stiefmeier, D. Roggen, G. Ogris, P. Lukowicz, and G. Tröster. Wearable activity tracking in car manufacturing. *IEEE Pervasive Computing*, pages 42–50, 2008.

[Stikic *et al.*, 2008a] M. Stikic, T. Huynh, K. Van Laerhoven, and B. Schiele. ADL recognition based on the combination of RFID and accelerometer sensing. In *Proceedings of* *the 2nd International Conference on Pervasive Computing Technologies for Healthcare*, pages 258–263, Tampere, Finland, January 2008. IEEE Xplore, IEEE Xplore.

[Stikic *et al.*, 2008b] M. Stikic, K. Van Laerhoven, and B. Schiele. Exploring semi-supervised and active learning for activity recognition. In *Proceedings of the 12th IEEE International Symposium on Wearable Computers (ISWC 2008)*, pages 81–88. IEEE, 2008.

[Subramanya *et al.*, 2006] A. Subramanya, A. Raj, J. Bilmes, and D. Fox. Recognizing activities and spatial context using wearable sensors. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*. Citeseer, 2006.

[Sun *et al.*, 2002] X. Sun, C. Chen, and B.S. Manjunath. Probabilistic motion parameter models for human activity recognition. In *Proceedings of the 16th International Conference on Pattern Recognition (ICPR'02)*, volume 1 of *ICPR '02*, pages 10443–, Washington, DC, USA, 2002. IEEE Computer Society.

[Sung *et al.*, 2005] M. Sung, C. Marci, and A. Pentland. Wearable feedback systems for rehabilitation. *Journal of NeuroEngineering and Rehabilitation*, 2:17, 2005.

[Suppes, 1970] P. Suppes. Probabilistic grammars for natural languages. *Synthese*, 22(1):95–116, 1970.

[Tapia *et al.*, 2004] E.M. Tapia, S.S. Intille, and K. Larson. Activity Recognition in the Home Setting Using Simple and Ubiquitous Sensors. *Lecture Notes in Computer Science*, pages 158–175, 2004.

[Tapia *et al.*, 2007] E.M. Tapia, S.S. Intille, W. Haskell, K. Larson, J. Wright, A. King, and R. Friedman. Real-time recognition of physical activities and their intensities using wireless accelerometers and a heart rate monitor. In *Wearable Computers, 2007 11th IEEE International Symposium on*, pages 37–40. IEEE, 2007.

[Vail *et al.*, 2007] D.L. Vail, J.D. Lafferty, and M.M. Veloso. Feature selection in conditional random fields for activity recognition. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2007. IROS 2007*, pages 3379–3384. IEEE, 2007.

[Van de Ven *et al.*, 2009] P. Van de Ven, A. Bourke, C. Tavares, R. Feld, J. Nelson, A. Rocha, and G. Laighin. Integration of a suite of sensors in a wireless health sensor platform. In *Sensors, 2009 IEEE*, pages 1678–1683. IEEE, 2009.

[Van Laerhoven and Cakmakci, 2000] K. Van Laerhoven and O. Cakmakci. What shall we teach our pants? In *The 4th International Symposium on Wearable Computers*, pages 77–83. IEEE, 2000.

[Van Laerhoven and Gellersen, 2004] K. Van Laerhoven and H.W. Gellersen. Spine versus porcupine: A study in distributed wearable activity recognition. In *8th International Symposium on Wearable Computers (ISWC 04)*, volume 1, pages 142–149. IEEE Computer Society, 2004.

[Van Laerhoven *et al.*, 2004] K. Van Laerhoven, B.P.L. Lo, J.W.P. Ng, S. Thiemjarus, R. King, S. Kwan, H.W. Gellersen, M. Sloman, O. Wells, P. Needham, et al. Medical healthcare monitoring with wearable and implantable sensors. In *Proceedings of the 3rd*

International Workshop on Ubiquitous Computing for Healthcare Applications. Citeseer, 2004.

[Villalba *et al.*, 2006] E. Villalba, M. Ottaviano, M. Arredondo, A. Martinez, and S. Guillen. Wearable monitoring system for heart failure assessment in a mobile environment. In *Computers in Cardiology, 2006*, pages 237–240. IEEE, 2006.

[Wang *et al.*, 2007] S. Wang, W. Pentney, A.M. Popescu, T. Choudhury, and M. Philipose. Common sense based joint training of human activity recognizers. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 2237–2242. Citeseer, 2007.

[Wang *et al.*, 2008] C. Wang, C. Chiang, P. Lin, Y. Chou, I. Kuo, C. Huang, and C. Chan. Development of a fall detecting system for the elderly residents. In *The 2nd International Conference on Bioinformatics and Biomedical Engineering*, ICBBE 2008, pages 1359–1362. IEEE, 2008.

[Wang, 1995] X. Wang. Learning by observation and practice: An incremental approach for planning operator acquisition. In *In Proceedings of the 12th International Conference on Machine Learning*, pages 549–557. Morgan Kaufmann, 1995.

[Ward *et al.*, 2006] J.A. Ward, P. Lukowicz, G. Tröster, and T. Starner. Activity recognition of assembly tasks using body-worn microphones and accelerometers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1553–1567, 2006.

[Weiser, 1995] M. Weiser. The computer for the 21st century. *Scientific American*, 272(3):78–89, 1995.

[Westeyn *et al.*, 2006] T. Westeyn, P. Presti, and T. Starner. ActionGSR: A combination galvanic skin response-accelerometer for physiological measurements in active environments. In *Wearable Computers, 2006 10th IEEE International Symposium on*, pages 129–130. IEEE, 2006.

[Wilson and Atkeson, 2005] D.H. Wilson and C. Atkeson. Simultaneous tracking and activity recognition (STAR) using many anonymous, binary sensors. In *The Third International Conference on Pervasive Computing*, pages 62–79. Springer, 2005.

[Witten *et al.*, 1999] I. Witten, E. Frank, L. Trigg, M. Hall, G. Holmes, and S.J. Cunningham. Weka: Practical machine learning tools and techniques with java implementations, 1999.

[Wojek *et al.*, 2006] C. Wojek, K. Nickel, and R. Stiefelhagen. Activity recognition and room-level tracking in an office environment. In *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pages 25–30. IEEE, 2006.

[Wong *et al.*, 1981] T.C. Wong, J.G. Webster, H.J. Montoye, and R. Washburn. Portable accelerometer device for measuring human energy expenditure. *Biomedical Engineering, IEEE Transactions on*, 28(6):467–471, 1981.

[Wyatt *et al.*, 2005] D. Wyatt, M. Philipose, and T. Choudhury. Unsupervised activity recognition using automatically mined common sense. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, pages 21–27. AAAI Press, 2005.

[Yang *et al.*, 2005] Q. Yang, K. Wu, and Y. Jiang. Learning action models from plan examples with incomplete knowledge. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005), Monterey, CA*, pages 241–250, 2005.

[Yin *et al.*, 2004] J. Yin, X. Chai, and Q. Yang. High-level goal recognition in a wireless LAN. In *Proceedings of the national Conference on Artificial Intelligence*, pages 578–584. AAAI Press, 2004.

[Zhang and Hartmann, 2007] H. Zhang and B. Hartmann. Building upon everyday play. In *CHI'07 extended abstracts on Human factors in computing systems*, pages 2019–2024. ACM, 2007.

[Zhang *et al.*, 2005] H. Zhang, L. Jiang, and J. Su. Hidden naive bayes. In *Twentieth National Conference on Artificial Intelligence*, pages 919–924. AAAI Press, 2005.

[Zhuo *et al.*, 2008] H. Zhuo, Q. Yang, D. Hu, and L. Li. Transferring knowledge from another domain for learning action models. *PRICAI 2008: Trends in Artificial Intelligence*, pages 1110–1115, 2008.

[Zhuo *et al.*, 2009] H.H. Zhuo, D.H. Hu, C. Hogg, Q. Yang, and H. Munoz-Avila. Learning htn method preconditions and action models from partial observations. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09)*, 2009.

[Zinnen *et al.*, 2007] A. Zinnen, K. Van Laerhoven, and B. Schiele. Toward recognition of short and non-repetitive activities from wearable sensors. *Ambient Intelligence*, pages 142–158, 2007.