Markovito's Team Description RoboCup@Home 2011

L. Enrique Sucar, Eduardo F. Morales, Blanca A. Vargas, Elva Corona, Irving Vásquez, Hussein López, Aarón Rocha, Adrián Leal, Manuel Oropeza, Pablo Oropeza, Patrick Heyer, and David Carrillo

National Institute of Astrophysics, Optics and Electronics,
Computer Science Department,
Luis Enrique Erro 1, 72840 Tonantzintla, México
{esucar,emorales,blanca,elvacx}@inaoep.mx
http://ccc.inaoep.mx

Abstract. The development of service robots has recently received considerable attention. Their deployment, however, normally involves a substantial programming effort to develop a particular application. With the incorporation of service robots to daily activities, it is expected that the they will require to perform different tasks. Fortunately, many of such applications share common modules such as navigation, localization and human interaction, among others. In this paper a general framework to easily develop different applications for service robots is presented. In particular, we have developed a set of general purpose modules for common tasks that can be easily integrated into a distributed, layered architecture, and coordinated by a decision–theoretic planner to perform different tasks. The coordinator is based on a Markov decision process (MDP) whose reward is set according to the task's goal, the states are represented by a set of variables affected by the general modules, and the actions correspond to the execution of the different modules. In order to create a new application the user only needs to define a new MDP whose solution provides an optimal policy that coordinates the different modules for performing the task. The effectiveness of our approach is experimentally demonstrated in four different service robot tasks with very promising results. Additionally, several of the modules include some novel ideas; in particular in navigation, localization and gesture recognition.

1 Introduction

Service robots are mobile robots that help people in different activities, such as helping elderly people in their home, serving as hosts and guides in museums or shopping malls, aiding in hospitals, etc. This idea of service robots to assist humans in every-day life has been around for many years. Although there is much work in developing different abilities for this kind of robots, little attention has been paid for the integration of these behaviors into a complete functional system. Such robots need different capabilities to perform their tasks, such as

navigation, mapping, localization and obstacle avoidance to move around in an uncertain and changing environment. They also need clear, simple and natural interactive interfaces to communicate with humans. In general, service robots are developed for executing a particular application, such as guides in a museum, however, it is desirable that a service robot could be capable of performing different tasks; or at least it should be *easy* to program it for other, similar, applications. Developing applications for service robots requires a considerable effort, however, most of them share a core of basic capabilities. It is then natural to develop different modules that can perform such common capabilities and combine them into a general architecture to relatively easy produce different applications.

In this paper we present a service mobile robot called Markovito. Markovito has the following main characteristics:

- **Generality**: it is based on different general–purpose modules for performing common tasks such as planning, navigation, localization, human tracking, object localization, voice interaction, etc. These modules are designed so they can be easily adapted and combined for different tasks.
- **Flexibility**: it is coordinated by a decision–theoretic controller that serves as an orchestra director of different general–purpose modules. New service robot applications can be easily constructed by defining a goal for the task, represented as a Markov decision process (MDP), and solving the MDP that coordinates different modules to perform this task.

Using this framework, Markovito has performed successfully different applications in a home environment. In particular it has solved the tasks of: (i) following a human under user commands, (ii) navigating to several places in the environment designated semantically, (iii) finding one of a set of different objects in a *house*, and (iv) delivering messages and/or objects between different people. The first three tasks are part of the RoboCup@Home challenge [25]. Experiments have shown that Markovito is reliable in the execution of these tasks, while keeping a simple and intuitive robot software architecture able to incorporate new abilities.

This chapter is organized as follows. Section 2 describes the software architecture used in Markovito. Section 3 describes different general software modules that can be used for different service robot's applications. Namely, map building and localization, navigation and planning, speech synthesis and recognition, and visual perception. In Section 4 the coordinator based on MDPs is presented. Section 5 describes the main implementation issues. Section 6 describes the different applications where Markovito has been tested. Finally, conclusions and future research directions are given in Section 7.

2 Software Architecture

Our software architecture is based on a Behavior-based architecture [1]. A behavior is an independent software module that solves a particular problem, such

as navigation or face detection. In this paper, behaviors are also referred as modules. Behaviors exist at three different levels:

Functional level: The lowest level behaviors interact with the robot's sensors and actuators, relaying commands to the motors or retrieving information from the sensors.

Execution level: Middle level modules perform the main functions, such as navigation, localization, speech recognition, etc. These interact with the lowest level through a shared memory mechanism. Each middle level module computes some aspect of the state of the environment. The outputs of these modules are typically reported to the highest level modules.

Decision level: The highest level coordinates the middle level modules based on a global planner. The planner consists of a Markov decision process that controls each module based on the goal (defined as a utility function) to perform a task.

The software architecture is depicted in figure 1.

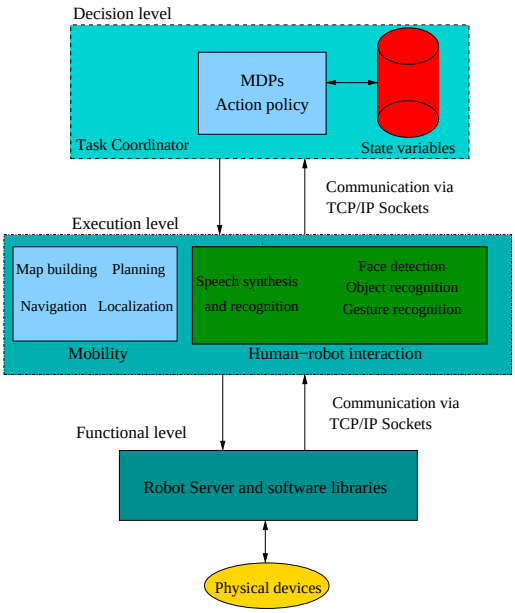


Fig. 1. Control Architecture.

This architecture can be implemented in a distributed platform, such that each level and each module within a level could be on a different processor. A transparent communication mechanism permits different configurations without need to modify the modules. Thus, some processing could be done on board the robot (lower level modules) and other off board (high level modules). Also a module can be changed without affecting the rest of the system.

The intermediate or execution level compromises the general–purpose modules that perform the main tasks for Markovito. Each module can receive different commands from the coordinator, which correspond to the *actions* of the MDP. Also, each module affects certain state variables, which are part of the *state* vector, used by the MDP to decide the *best action* according to its *policy*. The following section describe these modules, while Section 4 describes the coordinator.

3 Modules

We have implemented different general-purpose modules that are common to several service robot's applications, which are described in the following sections.

3.1 Map Building and Localization

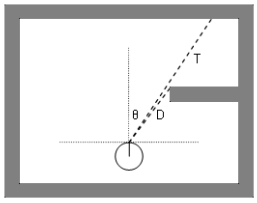


Fig. 2. An example of a discontinuity extracted from laser scanner data

A mobile robot requires a model or map of its environment to perform tasks. Our map building module uses information from a laser scan, its odometer and a sonar ring to construct an occupancy cells map using particle filters. Each particle represents a trajectory followed by the robot and a map associated with that path. The main idea is to represent the required posterior density function by a set of random samples with associated weights, and to compute estimates using those samples and weights. Increasing the number of samples increases the precision and approaches to the optimal Bayesian estimate [17]. To increase efficiency a Rao-Blackwellised particle filter approach [20] is normally used, that essentially marginalize some variables. The algorithm we have implemented is an extension of PMAP [12] that is able to work on line, includes information from laser as well as sonar readings, and have some parallel processing included to increase efficiency (see [13] for details).

The ability for mobile robots to locate themselves in an environment is not only a fundamental problem in robotics but also a pre-requisite to many tasks

such as navigation. There are two types of localization problems: local and global. Local localization techniques aim to compensate for odometric errors during navigation and require information about the initial position of the robot. Global localization aims to locate the robot's position without prior knowledge of its current location. These problems are particularly hard in dynamic environments where new obstacles can corrupt the robot's sensor measurements [8].

In order to locate itself either during navigation or globally, this module uses natural landmarks that have the advantage that the environment does not need to be transformed. In this work, discontinuities are used, that can be easily extracted from laser scanner data with high accuracy, to solve the local and global localization problem. A discontinuity is defined as an abrupt variation in the measured distance of two consecutive readings of the laser, as shown in Figure 2. Given a set of landmarks (discontinuities), a triangulation process is performed between all the visible landmarks to estimate the robot's position. The information from all the visible landmarks is combined considering the angle between landmarks, the distance between the robot and its farthest landmark, and if there are landmarks at both sides of the robot or only one, to give more accurate estimates.

For the global localization problem, a ray tracing approach is used to simulate laser readings on the map. Each cell is associated with all its visible landmarks and their values. This process is performed off-line once a map is constructed. To match a cell with the current readings of the robot, an initial stage filters out a large number of candidate positions with a fast algorithm. It counts the number of discontinuities obtained from the laser data that matches the distance, depth, and orientation of the previously stored discontinuities associated to each cell. A modified discrete relaxation algorithm is used in a second stage to determine the similarity of each cell with the observations of the robot, considering the distances between the discontinuities in this stage. Our global localization algorithm is able to locate the robot even with new obstacles as can be seen in Figure 3, where five new obstacles are added to the environment (see [6] for more details).

3.2 Navigation

There are different strategies that can be applied for navigation. For instance, we have implemented a navigation module that uses a dynamic programming algorithm, with exponential costs near obstacles, to find the least expensive path. In order to avoid new obstacles, the robot is sensing its environment while moving. In case a new obstacle is placed in front of the robot the module finds an alternative path, as shown in Figure 4 (see [10] for more details).

In this paper, a novel navigation strategy using machine learning techniques is presented. To illustrate more clearly the motivation behind this work, imaging going to a new place, e.g., a conference site. You would normally ask for

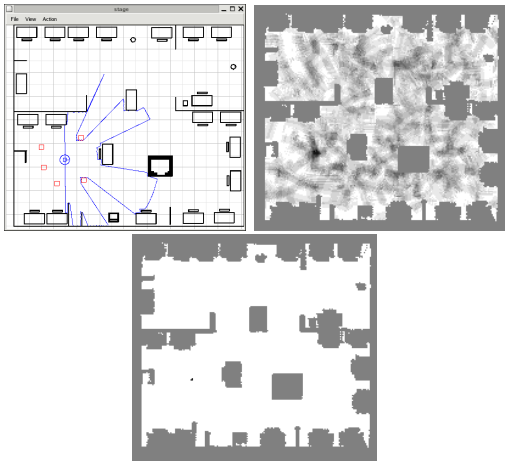


Fig. 3. Global localization. Top left: a simulated environment, showing the robot and some discontinuities. Top right: results of the 1st. stage. Bottom: the robot is localized after the 2nd. stage

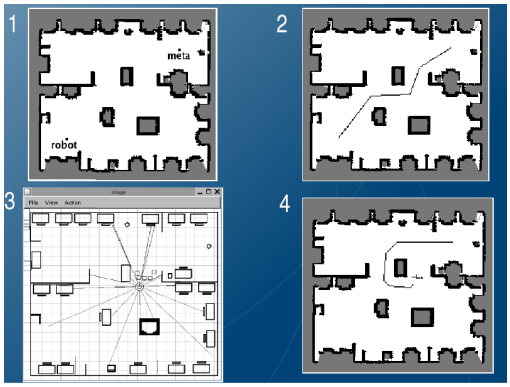


Fig. 4. Re-planning with obstructed paths.

directions of places of interest, like the registration desk or a toilet, and you will get general directions, like "at the end of the aisle to your right" or possibly "in room 203". You will have then to navigate without collisions in an unknown and dynamic environment to a particular destination point through well known natural marks like walls and doors and expected dynamic conditions, like walking people. Imagine you want your robot to learn how to perform a similar skill. You place your robot in an unknown environment and you want it to navigate to a particular point, like a charging station, but the robot is only given the general direction of its destination point. The robot has to learn how to perform simple skills, like obstacle avoidance and orientation towards a goal, and use them to

safely go to a particular goal in a dynamic and unknown environment. This is the approach follow in this paper for the navigation module.

The application of machine learning techniques with expressive representation languages to domains like robotics have received little attention due to the huge amount of low level and noisy data produced by the sensors. The use of relational representations in this area are novel, and their advantages have just recently being addressed (e.g., [9]). In this paper, first-order logic relations are learned to build the navigation module. This module consist of a set of reactive rules that sense the environment continuously and apply actions whose continuous execution eventually satisfy a goal condition. These rules are known as TOPs (Teleo-OPerators) [5], an effective framework to achieve goals when unexpected events occur.

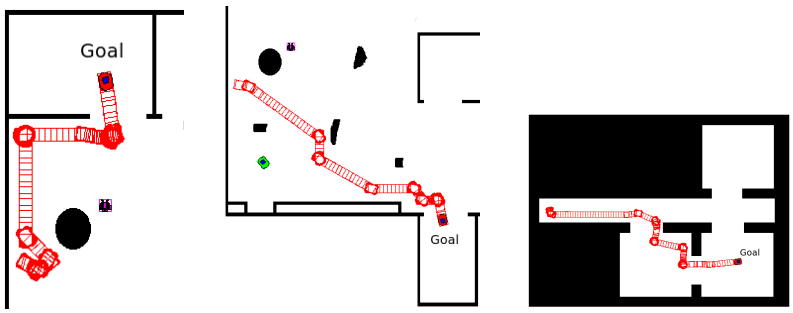
The objective of our learning process is to provide a mobile robot with abilities to move through indoor environments and to accomplish goals. In order to learn TOPs, we combine three machine learning techniques: (i) behavioural cloning, (ii) inductive logic programming (ILP), and (iii) a simple grammar learning algorithm. Behavioural cloning is a technique to learn skills from examples [18]. The key idea of this method is to show the robot what to do instead of how to do a task. For instance, in order to learn how to navigate without collisions, the robot is presented with human traces steering the robot avoiding obstacles, simplifying the programming effort.

We introduced a two phase learning process to learn TOPs for mobile robots in indoor environments: (i) learning of basic TOPs, and (ii) learning of complex TOPs. In the first phase, a system called TOPSY uses human-guided traces and the natural landmark identification process, described in the previous section, to reduce the information from the sensors into a small set of ground predicates (landmarks) suitable for an ILP system called ALEPH [28]. A small set of background knowledge is given to the system from which other predicates are learned and used in the induction of simple TOPs.

TOPSY was able to learn the following TOPs from human traces:

- *avoid(State,Action)*: to wander around without collisions. *Action* can take the following values: *go_forward*, *turn_left*, and *turn_right*.
- *orient(State,Action)*: Given a target point, the robot has to turn until it is oriented towards the goal, only if it is located in a *safe_turn* zone. *Action* can take the following values: *turn_right*, *turn_left* and *nothing*.

In order to learn how to combine TOPs to perform more complex tasks (e.g., the goto TOP), previously learned TOPs that apply to states in traces are identified, returning high-level traces of applicable TOPs. The goal is to learn a grammar with TOPs able to reproduce a set of traces. With this purpose, FOSeq (First Order learning from Sequences), an algorithm to learn grammars from sequences based on association rule learning is introduced and applied to induce the goto TOP. This algorithm is in its initial stage, however, the learned TOPs were used for navigation tasks on different environments with dynamic objects. Figure 5 shows some examples of the *Goto* TOP under different scenarios.



(a) Round obstacle (b) Several obstacles (c) Goto in a corridor

Fig. 5. Goto in different scenarios

3.3 Planning

The previous TOPs are used with a probabilistic roadmap module that returns collision free paths. A probabilistic roadmap (PRM) [3, 15] is build using a random generation of points in the configuration space. This points are joined if there is a free path between them and the information is stored in a graph G. Given an initial configuration s and a goal configuration g, the idea for path planning is to connect s and g to nodes s' and g' in G. Given the random nature of the algorithm, some paths in G may be too long and irregular, so a smoothing algorithm, trying to find shortcuts, is applied. This process is illustrated in figure 6. A combination of the roadmap points with the TOPs produces a robust navigation module.

3.4 Speech synthesis and recognition

One of our main initial goals was to provide Markovito with the ability to speak and understand Spanish language. Among the advantages of this approach we can cite the possibility to test our robot with a greater number of people with different backgrounds. Markovito is able to synthesize many pre-defined phrases in Spanish language. In our work, we identified that the robot needs to talk with its user in order to: i) notify when it is ready to initiate an interaction with a user (e.g., receive an instruction), ii) request some aid or information in order to effectively complete the task at hand (e.g., to ask for the name of the current user), and iii) inform when the task has been completed. In this way, many of these phrases are recycled for the different tasks developed by this robot. Table 1 shows some examples of Markovito's phrases. For some tasks, several sentences with the same meaning were defined and one of them was randomly selected during interaction to produce a more realistic conversation with the user.

Speech recognition is based on the *Dimex* project [22]. This work is aimed to develop general acoustic-phonetic models and pronouncing dictionaries for Mexican Spanish language inside the *Sphinx2* framework for continuous and multi-

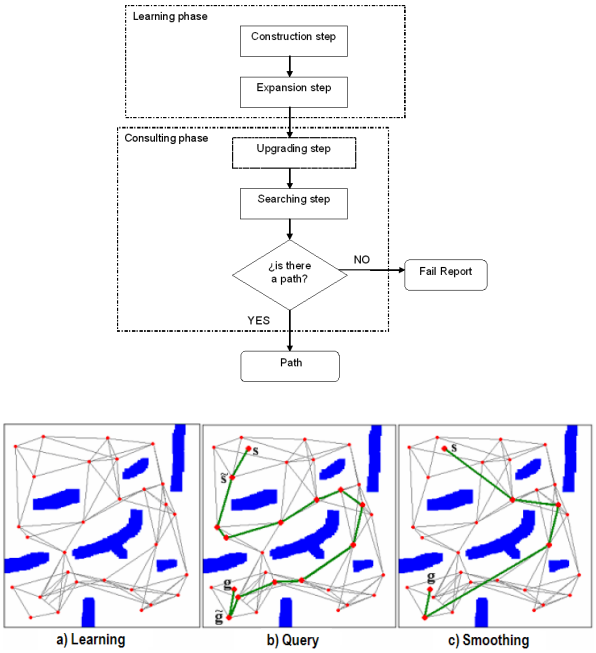


Fig. 6. Probabilistic road maps (PRMs). The top figure shows a simplified diagram of the algorithm with the two main phases: learning and consulting. The top figure depicts the 3 stages in the construction of the path from the origin to the goal.

Table 1. Some examples of the phrases Markovito can synthesize

Phrases in Spanish	Translation to English
Esperando orden	Waiting for a command
Por favor, dime tu nombre	Please tell me your name
Por favor deme el mensaje	Please give me the message
Por favor, coloque el objeto en la pinza	Please place the object into the gripper
Gracias por usar mis servicios	Thanks for using my services
...	...

speaker recognition[1]. The speech recognition system works with male and female speakers. Unfortunately, during the initial development cycle, this approach obtained poor results. At the same time: i) people had to talk very close to the Peoplebot robot's original twin-microphones in order to be heard by the robot and ii) the recognition step was highly affected by the environment sounds (e.g., other conversations) and the robot itself (e.g., robot's sonars, etc.). To cope with these problems it was decided to incorporate a directional microphone SHURE model SM81 to the robot. With this microphone a person can talk the robot at a distance that ranges from 30 cm. to one m. However, the environment noise still was a problem. In order to eliminate some noise frequencies from the sound signal before the recognition process, we filter this signal with a noise removal filter adapted from the Audacity's open source code[2]. In order to enhance the signal, this was amplified using an adaptive approach. These simple modifications resulted in an improvement with good results for our purposes.

3.5 Visual Perception

Service robots must integrate visual abilities such as people detection, recognition of their activities and object recognition in order to interact effectively with its users and its environment. In this section we discuss the visual capabilities implemented on our service robot. These features include face detection, object detection and gesture recognition.

Face detection Face detection is based on the AdaBoost algorithm proposed by [31]. The main idea of this algorithm is the linear combination of classifiers for face features such as eyes, mouth, and nose. Under this scheme, instead of using a single classifier for the whole face, they propose the use of a combination or *cascade* of simple classifiers. The detection performance of AdaBoost is competitive in comparison to similar approaches [26] while decreasing the required processing time.

Object recognition Object recognition is carried out using the SIFT or Scale-Invariant Feature Transform algorithm [16]. This algorithm aims to detect and describe distinctive features of the objects. As its name says, features are invariant to scale, translation, and rotation. This is due to the extraction process that search stable features of the original image at different scales. The algorithm proceeds as follows. First, the original image is convolved with various Gaussian filters (by varying σ) at different scales to obtain a set of blurred images grouped by scale. Then, the difference (difference of Gaussians, $DofG$) from adjacent blurred images of the same scale is computed. From this new set of DofG images a set of features are extracted. These features can be stored for a future match with features extracted from incoming images to recognize an object or a scene.

[1] Available at: http://cmusphinx.sourceforge.net/html/
[2] Available at: http://audacity.sourceforge.net/

In Markovito, a data base of objects is created by showing to the robot a particular object, whose distinctive features are obtained and stored. Later, the robot can be ask to look for any of the objects that were previously shown.

Gesture recognition Gesture recognition is a key element on natural human-robot communication. In our work, we propose to approach this problem by integrating posture and motion information to characterize a set of 9 gestures (Fig. 7). This scheme allows us to increase recognition rates significantly, even when considering gestures with similar motions.

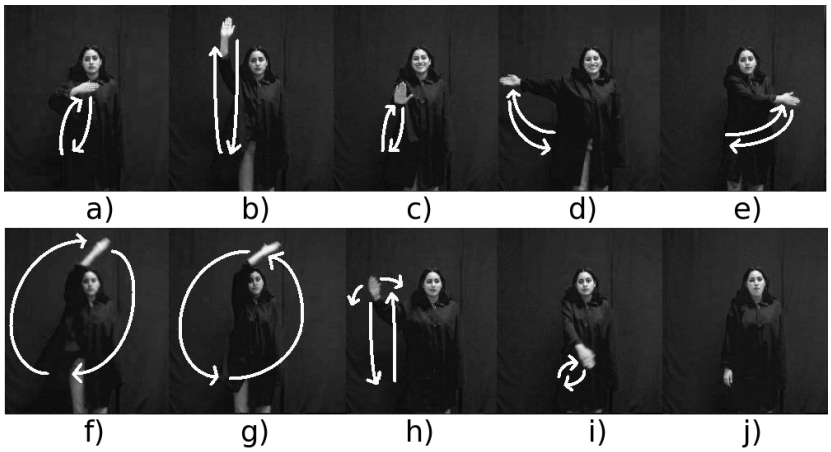


Fig. 7. Gestures considered by our system: a) come, b) attention, c) stop, d) go-right, e) go-left, f) turn-left, g) turn-right, h) waving-hand and i) pointing; j) shows the initial and final position for each gesture.

To recognize gestures we propose an alternative model for hidden Markov models (HMMs), that we call *dynamic naive Bayesian classifiers* (DNBCs) [2]. A DNBC is composed by: i) a set $C = \{C_t | t = 1, \ldots, T\}$, where each C_t is a random variable that can take one of N possible classes at each time t, and ii) a set $A = \{A_t | t = 1, \ldots, T\}$, where each $A_t = \{A_t^1, \ldots, A_t^M\}$, is a set of M instantiated random variables generated by the process at time t.

The main difference between the DNBCs and HMMs probability functions is that the attributes are assumed independent given the class or state variable. This factorization: i) enables us to consider techniques to explore statistical relationships among attributes, ii) reduces the number of parameters, so it could also reduce the number of training samples required, and iii) augments the clarity of the graphical representation of the model. Figure 8 shows a DNBC unrolled three times with three attributes.

Training equations for DNBCs can be derived in a similar manner as it is for HMMs [24]. For recognition, maximum-likelihood (ML) criterion can be used to

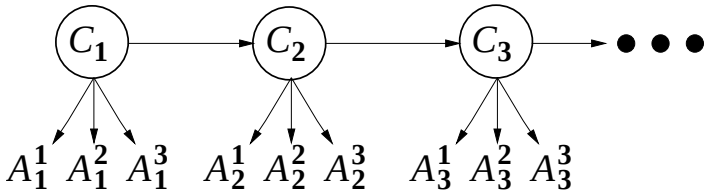


Fig. 8. Graphical representation of a dynamic naive Bayesian classifier.

select the model that maximizes $P(A|\cdot)$. In comparison to HMMs, DNBCs requires less iterations of the EM algorithm for training, while keeping competitive classification rates.

To capture our gestures we implemented a monocular visual system using OpenCV libraries for Linux. The system starts with a person standing in a rest position, at a distance of about 3 m. in front of the video camera. Face detection is carried out by using the Adaboost algorithm proposed in [30]. The position of the right-hand and torso regions are estimated using anthropometrical measures based on face dimensions. To segment the hand by skin color different lighting conditions of various users, we developed an adaptive scheme by combining a *general*, $Pg(rgb|\cdot)$, probability distribution constructed off-line, with a *personal* one, $Pp(rgb|\cdot)$, created on-line by sampling randomly the face and torso of the user. Those functions are combined by means of the next rule:

$$P(rgb|\cdot) = Pg(rgb|\cdot) \times Pp(rgb|\cdot).$$

In this way, one pixel is classified as skin if $P(rgb|s) > P(rgb|\neg s)$. This strategy allows the visual system to track the hand accurately in several environmental conditions. Once the initial position of the hand is estimated, the CAMSHIFT algorithm [7] is used to track the hand motion over the rest of the image sequence. Our visual system is able to process up to 30 f.p.s. (using an IBM PC Intel Pentium 1.6Ghz, 512Mb RAM), although we sampled attributes at a lower rate. The image resolution is 640480 pixels[3]. See [2] for more details of this work.

Next the top–level coordinator, that directs the different software modules, is described.

4 Coordinator

4.1 Markov Decision Processes

Markov decision processes (MDPs) have become the semantic model of choice for decision theoretic planning (DTP) in the artificial intelligence community [23].

[3] A video that shows the application of this recognition system for the teleoperation of a mobile robot can be found here: http://www.youtube.com/watch?v=opAUo0zJHGY.

They are simple for domain experts to specify, or can be learned from data. They have many well studied properties including approximate solution and learning techniques. An MDP is a tuple $\{\mathcal{S}, \mathcal{A}, \Pr, R\}$, where $\mathcal{S}$ is a finite set of states and $\mathcal{A}$ is a finite set of actions. Actions induce stochastic state transitions, with $\Pr(s, a, t)$ denoting the probability with which state t is reached when action a is executed at state s. $R(s, a)$ is a real-valued reward function, associating with each state s and action a. Solving an MDP is finding a mapping from states to actions. Solutions are evaluated based on an optimality criterion such as the expected total reward. An optimal solution is one that achieves the maximum over the optimality measure, while an approximate solution comes to within some bound of the maximum.

A solution to an MDP is a policy that maximizes its expected value. For the discounted infinite–horizon case with any given discount factor $\gamma \in [0, 1)$, there is a policy V^* that is optimal regardless of the starting state that satisfies the *Bellman* equation [4]:

$$V^*(s) = max_a\{R(s, a) + \gamma \Sigma_{s' \in S} \Phi(a, s, s')V^*(s')\} \quad (1)$$

Two popular methods for solving this equation and finding an optimal policy for an MDP are: (a) value iteration and (2) policy iteration [23].

The space and time complexity of MDPs increases with the number of states. This problem can be reduced by using factored representations [14], in which the state is decomposed in a set of variables or factors, and the transition functions is represented using a factored representation (dynamic Bayesian nets). In this paper we use a factored representation to specify the MDPs and SPUDD [11] to solve them. SPUDD uses the value iteration algorithm to compute an optimal infinite-horizon policy of action for each state, with *expected total discounted reward* as the optimality criterion. Ir uses a representation of MDPs as decision diagrams, and is able to take advantage of structure in the underlying process to make computation more efficient and scalable towards larger environments. The modularity of our system makes representation as a factored MDP simple and typically results in a sparsely connected Markov network. Such sparseness leads to very efficient calculations when using a structured solution approach as in SPUDD.

4.2 Task Coordination based on MDPs

To apply MDPs for coordinating a task for Markovito, we have to define the model and then solve it using SPUDD. The model compromises the following elements:

- The global state of the system described by a vector **S**. These is the set of high level variables that are relevant for the task. The coordinator synthesizes this state vector by collecting information from each module.
- The set of high level actions, **A**, required to complete the task. These actions are implemented by calling the different modules with the corresponding parameters.

- The goal of the task defined as a reward function, **R**, that specifies high positive values for the desired states and negative values for the states that should be avoided.
- The transition function $\Pr(s, a, t)$.

This model is currently specified manually by the programmer according to the task. In general it is relatively *easy* to specify this model, in particular the state variables, actions and rewards. It is more complicated to specify the transitions function; however, once given for a task, they are in general similar for other tasks in the same or similar environments. We use an iterative approach to define the model. An initial model is defined and solved with SPUDD. Then we use a simulator to verify the obtained policy, and if there are inconsistent or *strange* actions, the model is modified and the process is repeated. For the tasks developed so far, it took less than one week to have develop each model. Based on the final model, the optimal policy is obtained, and this is used to coordinate all the software modules to perform the task. The model is specified and solved off-line, and just the final policy is used on-line. A simple program, that we call *manager*, implements the optimal policy (stored in a table), by reading the state vector and selecting the optimal action, which is directed to the corresponding module.

We are assuming that Markovito knows with certainty its state, but in reality there is uncertainty, so we might consider instead the belief state. We are currently compressing the probabilistic belief states reported by the modules by choosing the value with maximum likelihood. In the general case, the planners would take advantage of the information contained in the belief state, by using partially observable MDPs (POMDPs). However, it is P-SPACE hard to find policies for POMDPs, calling for approximate techniques for robotics applications [19]. Hierarchical methods are another way to combat the complexity of POMDPs [27, 29, 21]. Markovito considers the state vector to be fully observable, which is a good approximation when the uncertainty in not too *high*. That is, when the probability of the most probable state is significantly higher than other states (the entropy is low). As future work we propose to extend our approach to POMDPs.

The combination of coordinator based on MDPs with a 3-level architecture can reduce the development costs of different service robots applications. By simply re-arranging the modules, changing the goal (reward), and solving a new MDP, different applications can be developed. Different software modules of common tasks can be incorporated in a transparent way. In the following section we describe Markovito's hardware and implementation issues.

5 Implementation Issues

In this section we describe the hardware and software platform used for Markovito and the general structure followed for its implementation. We believe this kind of explanation is important because in the literature it is common to present the

architectural design of the system only, ignoring almost completely the implementation details, which are very useful to developers.

5.1 Hardware and Software Platforms

Markovito is based on a PeopleBot robot platform (ActivMedia). It has two rings of sonars, a Laser SICK LMS200, one video camera Canon VCC5, two infrared sensors, a directional microphone SHURE SM81, a gripper, the robot's internal computer, a Laptop, bumpers and a frame grabber WinTV USB 2 (see Figure 9).

Fig. 9. Markovito.

Table 2 summarizes the software libraries and source code considered to develop each module. As it is shown in Figure 1 each module was independently implemented. This gave us the facility to: i) test each module separately (even on different computers), ii) in some cases to use simulation (e.g., navigation, localization, map construction and object grasping behaviors), and iii) speed up the development and testing processes.

Table 2. Modules and libraries used in Markovito.

Module	Source code/Libraries
NAVIGATION	Player/Stage server, pmap utility, sw-prolog
VISION	Player/Stage server, OpenCV, SIFT algorithm
DEVICES	Player/Stage server
SPEECH	Sphinx2, Audacity, Listener, Festival Player/Stage server
MDP	SPUDD

5.2 General structure and operation

In our design, each module has assigned a set of state variables and actions that it can modify and execute, respectively. The system starts by running each module first, and lastly the MDP module. Once this has be done, the MDP immediately tries to send the corresponding variables and their values to each module. This operation serves as an initialization step for the variables stored in the modules. When the MDP has received in response an acknowledgment from the modules, then it knows that everything is working fine and it is time to send the first action – from the policy obtained off-line with SPUDD.When a module receives an action, it proceeds to execute it. In all cases our implementation design considers that the action is performed on its own thread, enabling the module to receive a new command if it is necessary. In this case, the module can: i) abort the current action and start the new one, ii) queue the second one or, iii) ignore it, if it is desired. This could be useful when considering parallel actions – the same action executed at the same time for two or more modules.

The information exchange between modules is carried out only through the MDP. The communication between each module and the MDP is done via client-server tcp/ip sockets. MDP considers 2 threads per module, one to write to and another to read from each module. The later one selects and sends actions once the value of a new variable has been received from a given module. This enable the MDP to send and receive actions and variables asynchronously. Currently Markovito has computers, the internal one and a laptop mounted on top the robot. The communication between these two machines is done through an Ethernet 100Mb/s cable that connects the laptop and the robot; in this form we: i) avoid possible problems with wireless networks that are usual to occur in practice and ii) decrease the delay time of the communication that is slower when using wireless networks. This is important in some contexts and to enhance autonomy for the robot. Moreover, by simply changing the line-command parameters of each module we can run the system on the laptop computer or in the robot's computer.

The complete system was developed using C/C++ language mainly (except for the TOPs rules that used sw-prolog). A video that shows the system running can be found at: http://videolectures.net/aaai07_aviles_srm/.

6 Applications

This section presents the different tasks Markovito is able to perform. These tasks are based on those proposed for the RoboCup@Home competition[4]. As mentioned in Section 4, the different modules are coordinated with an MDP.

[4] http://robocup.org

6.1 Navigate through three places and return to the starting position

In this task, Markovito has to navigate safely through three places, that are given by a user at the beginning of the interaction, and then return to its starting position. As soon as the robot reaches a destination point it has to announce it. All the interaction is made through natural language.

Six variables were defined for this MDP: *localize* indicates when the robot's position is know, *get goals* when the robot has obtained the three places to visit, *arrived at destination point*, *has trajectory* for navigation, *end position* when it returns to the starting position. The MDP has six actions to coordinate the modules which are described in Table 3. Each action calls one of the previously described module.

Table 3. The six actions used to coordinate the modules for the navigation task

Action	Module	Description
Localize	Navigation	Global localization
Wait for a goals	Speech	Obtain the places to visit
Generate trajectory	Navigation	Give a path to a destination place
Go next goal	Navigation	Navigate to reach the next place
Announce arrival at ith. goal	Speech	Announce that the robot has reached a new place
Announce finish	Speech	Announce that the robot has returned to the initial position

Figure 10 shows a sequence of Markovito navigating through the environment reaching different destination points.

6.2 Lost & Found

In this task, Markovito is shown an object it which later has to search for in the environment. The user places several objects, one by one, in front of the robot's camera. Afterwards, the user tells the robot which object to search for and a destination area. Markovito has to navigate to the indicated area and search for the object. It sends a recognition message when the object is recognized.

Markovito's first action is to find its global position in the environment. Once this has been done, the robot notifies that it is ready to accept a user instruction. We considered three different commands: i) learn an object, ii) search an object and iii) forget an object. If the robot receives a command to search or forget an object before knowing it, a notification is given to the user. We included the forget instruction just in case of any mistake when positioning the object in front of the robot's camera. Once at least one object has been presented to the robot,

Fig. 10. Markovito navigating through the environment reaching different destination points

the user can instruct the robot to search for the object. This instruction can be complemented with the name of a place in the environment (e.g., kitchen, table, etc.) where the robot can go to search the object. When the robot has reached the given place, it starts to pan and tilt its camera to capture images of the environment. These images are processed using the SIFT algorithm to detect the object.

This MDP has 11 variables: *localize, has order, forget object, confirm forgetting, learn object, confirm learning, object in data base, found, reached destiny, is near*, and *confirm object*. The actions consider in this MDP are: *global localization, wait for order, forget object, learn object, confirm learn object, confirm forget object, go to search area, look for object, get close to object*, and *confirm found object*. All the interactions in this task are in natural language. This task uses the localization, navigation, planning, speech and perception modules. Figure 11 shows Markovito learning an object, in this case a Teddy bear (left) and then searching for it (right).

Fig. 11. Markovito learning to recognize an object (left) and later searching for it (right)

6.3 Follow a person under user request

For this task, Markovito has to follow a human through an unknown track in a home-like environment. After reaching the end position, the robot has to return to its starting position. The task consisted of two stages. In the first stage, the human stands in front of the robot at a distance of one meter for about one minute for calibration. At this state a torso detection module was used. In a second stage, the human starts walking towards the end position, passing through a number of places. This task used the navigation module based on TOPs to go dynamically to changing places given by the torso tracking module.

This MDP has 13 variables: *localize, person, calibrate order, follow order, stop order, calibrate, follow, searching, announce following, announce searching, announce finish, has trajectory*, and *get destination*. All the interaction is also in natural language. Table 4 shows the actions that are used to coordinate the different modules.

Table 4. The actions used in the task to follow a person and the different modules used in this task

Action	Module	Description
Localize	Navigation	Global localization
Wait order to calibrate	Speech	Wait for a calibration order
Calibrate	Vision	Get information about the person
Wait order to follow	Speech	Wait for a follow order
Follow	Navigation	Follow a person
Announce searching person	Speech	Announce that the robot lost the person
Search the person	Vision	Search for a person
Announce found person	Speech	Announce that the robot found the person
Wait order to stop	Speech	Wait for a stop order
Generate trajectory	Navigation	Give a path to destination place
Return	Navigation	Navigate to the starting position
Announce arrive	Speech	Announce arriving to the starting position

6.4 Deliver messages and objects between people

Markovito is able to act as a messenger robot to deliver spoken messages, objects, or both under a user's request. Markovito first obtains its global position and orientation (x, y, θ) relative to its environment[5]. Once it has found its global position, Markovito waits for the spoken salute of a user and when it arrives, the interaction starts. At this stage the robot asks the name of the sender and receiver, and requests the message or object, or both. Then, the robot proceeds to navigate to the receiver's position – assumed known in advance by the robot. At this place Markovito tries to detect the face of a user. If a face is detected, Markovito delivers the message or the object and resets the state variables to their default values waiting for a new request.

An MDP with 12 binary variables was used for this task: *localized, greetings, sender, receiver, message, object, trajectory, reached destiny, found receiver, object/message delivered, message*, and *object*. Two multi-valued variables were also used for the battery level and for the type of delivery. Thirteen actions were defined for this task: *localize globally, generate path to destination place, execute trajectory*, as part of the navigation module. *Wait for greeting, request type of delivery, request sender's name, request receiver's name, record message, deliver message, confirm delivery*, as part of the speech module. *Grasp object, release object* as part of the delivery module. *Detecting a person* is part of the vision module. Figure 12 shows Markovito delivering a beer to a person watching T.V.

Fig. 12. Markovito delivering a beer

[5] It is assumed that a map was previously built as described in section 3.1 and uses natural landmarks for its global localization.

7 Conclusions and Future Work

A general framework for creating service robots applications has been described. This framework uses different modules to perform common service robot tasks. We have implemented different modules that include some novel ideas, like natural landmarks based on discontinuities for localization, machine learning techniques for inducing TOPs for navigation, and dynamic naïve Bayesian classifiers for gesture recognition. The architecture uses an MDP framework to coordinate these modules. Different service robot applications can be easily constructed using these modules and defining a new MDP to obtain an optimal policy for each task. We have illustrated the capabilities of this framework with different applications.

There are several research directions to follow. In particular, we plan to incorporate a module that can be used to teach the robot by example, to incorporate a face recognition module, and to combine gesture recognition with the natural language module to enhance our human-robot interface. We are also improving the coordinator to deal with conflicting situations. Finally, we plan to use this framework for developing other applications.

References

1. R.C. Arkin. *Behavior-based Robotics*. MIT Press, 1998.
2. H. Aviles and L.E. Sucar. Recognizing similar gestures. In *Proc. IEEE Inter. Conf. on Pattern Recognition (ICPR)*, China, August 2006.
3. J. Barraquand and J. C. Latombe. Robot motion planning: A distributed representation approach. *Journal of Robotics Research*, 1991.
4. R.E. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, 1957.
5. Scott Benson and Nils J. Nilsson. Reacting, planning, and learning in an autonomous agent. *Machine Intelligence*, 14:29–62, 1995.
6. A. BlindReview. Global localization of mobile robots for indoor environments using natural landmarks. In *Proceedings IEEE Conference on Robotics, Automation and Mechatronics (RAM-2006)*, 2006.
7. G.R. Bradski. Real time face and object tracking as a component of a perceptual user interface. In IEEE Computer Society, editor, *Proceedings of the 4th IEEE Workshop on Applications of Computer Vision (WACV'98)*, pages 214–219, 1998.
8. W. Burgard, A.B. Cremers, D. Fox, D. Hahnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. The interactive museum tour-guide robot. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI '98)*, Madison, Wisconsin, July 1998.
9. A. Cocora, K. Kersting, C. Plagemann, W. Burgard, and L. De Raedt. Learning relational navigation policies. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Beijing, China, 2006.
10. S. Hernández. Navegación de un robot móvil en ambientes interiores usando marcas naturales del ambiente. Master's thesis, Instituto Tecnológico y de Estudios Superiores de Monterrey - Cuernavaca, 2005.

11. J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. Spudd: Stochastic planning using decision diagrams. In *Proc. of the 15th Conf. on Uncertainty in AI, UAI-99*, pages 279–288, 1999.
12. A. Howard. Simple mapping utilities (pmap), 2004.
13. V. Jáquez. Construcción de mapas y localización simultánea con robots móviles. Master's thesis, Instituto Tecnológico y de Estudios Superiores de Monterrey - Cuernavaca, 2005.
14. L.P. Kaelbling, M.L. Littman, and A.R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2), 1998.
15. J. C. Latombe. *Robot motion planning*. Kluwer Academics Publishers, 1991.
16. D. Lowe. Distint image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.
17. S. Maskell and N. Gordon. A tutorial on particle filters for on-line nonlinaer/non-gaussian bayesian tracking. In *IEEE Colloquim on Tracking*. 2002.
18. Donald Michie and Claude Sammut. Behavioral clones and cognitive skill models. *Machine Intelligence*, 14:395–404, 1995.
19. M. Montemerlo, J. Pineau, N. Roy, S. Thrun, and V. Verma. Experiences with a mobile robotic guide for the elderly. In *Proceedings of the AAAI National Conference on Artificial Intelligence (AAAI '02)*, Edmonton, Canada, 2002.
20. K. Murphy and S. Russell. Rao-blakwellised particle filtering for dynamic bayesian networks. In *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, 2001.
21. J. Pineau, G. Gordon, and S. Thrun. Policy-contingent abstraction for robust robot control. In *Proceedings of Uncertainty in Artificial Intelligence (UAI)*, pages 477–484, Acapulco, Mexico, August 2003.
22. A. Pineda, L. Villase nor, J. Cuétara, H. Castellanos, and I. López. Dimex100: A new phonetic and speech corpus for mexican spanish. In C. Lemaitre, C.A. Reyes, and J.A. González, editors, *Advances in Artificial Intelligence, Iberamia-2004, Lectures Notes in Artificial Intelligence 3315*, pages 974–983, 2004.
23. M.L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, New York, NY., 1994.
24. L.R. Rabiner. Tutorial on hidden markov models and selected applications in speech recognition. In *Readings in Speech Recognition*, pages 267–296. Morgan Kaufmann Publishers, 1990.
25. RoboCup. The robocup@home webpage.
26. H.A. Rowley, S. Baluja, and T. Kanade. Human face detection in visual scenes. Technical Report CMU-CS-95-158R, School of Computer Science, Carnegie Mellon University, 1995.
27. R. Simmons and S. Koenig. Probabilistic robot navigation in partially observable environments. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI '95)*, pages 1080–1087, Montreal, Canada, 1995.
28. Ashwin Srinivasan. The aleph 5 manual http://web.comlab.ox.ac.uk/oucl/research/areas /machlearn/ aleph. 2005.
29. G. Theocharous, K. Rohanimanesh, and S. Mahadevan. Learning hierarchical partially observable Markov decision process models for robot navigation. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA '01)*, Seoul, Korea, May 2001.
30. P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR '01)*, 2001.
31. P.A. Voila and M.J. Jones. Robust real-time object detection. *International Journal of Computer Vision*, pages 137–154, 2001.