Robots in the kitchen: Exploiting ubiquitous sensing and actuation

Radu Bogdan Rusu⁠, Brian Gerkey, Michael Beetz

A R T I C L E   I N F O

Article history:
Available online 3 July 2008

Keywords:
Ubiquitous robotics
Sensor network
Software infrastructure
Reusable code

A B S T R A C T

Our goal is to develop intelligent service robots that operate in standard human environments, automating common tasks. In pursuit of this goal, we follow the ubiquitous robotics paradigm, in which intelligent perception and control, are combined with ubiquitous computing. By exploiting sensors and effectors in its environment, a robot can perform more complex tasks without becoming overly complex itself. Following this insight, we have developed a service robot that operates autonomously in a sensor-equipped kitchen. The robot learns from demonstration, and performs sophisticated tasks, in concert with the network of devices in its environment. We report on the design, implementation, and usage of this system, which is freely available for use, and improvement by others, in the research community.

© 2008 Elsevier B.V. All rights reserved.

1. Introduction

We aim to develop intelligent service robots that operate in standard human environments, automating common tasks. Today, the research community has focused primarily on self-contained, stand-alone robots that would act autonomously in unmodified environments. The goal is to enable a robot to do, like humans and other animals do, all sensing, deliberation, and action selection on board. We advocate an alternative path to competent robotic agency, known as ubiquitous robotics, that combines intelligent perception and control with ubiquitous computing [29, 23].

Computing is ubiquitous when computing devices are distributed and embedded invisibly into the objects of everyday life. These devices sense their environment, connect automatically to each other to form sensor networks, exchange information, and act to modify their environment. They range in complexity from simple, embedded sensors, to traditional autonomous mobile robots. For example, in a sensor-equipped kitchen, cupboards “know” what is inside them because objects are tagged with RFID (Radio Frequency Identification) tags, and cupboards are equipped with RFID tag readers. A robot whose task is to deliver coffee mugs could benefit greatly from access to information about the cupboards’ contents.

If we consider the future of service robotics, it seems likely that service robots will be competent, and versatile agents in sensor- and effector-equipped operating environments, rather than autonomous and insular entities. This is the basic idea of ubiquitous robotics. Following this paradigm, a robot can connect to the sensing and actuation network of its operating environment, and use the sensors and actuators, as if they were its own.

Ubiquitous robotics is a promising route to achieving autonomous service robots, because sensor and computer networks can substantially enhance the robots’ perceptual and actuation capabilities in important ways.

- Special purpose sensors. Rather than relying on general purpose sensors such as cameras and laser scanners, sensor networks allow for the definition of task-specific sensors. Using RFID tags and readers and acceleration sensors for objects and hands, sensor networks can detect force-dynamic events such as an object being picked up or put down. Or, using long range RFID tag readers in cupboards and under tables, the network can sense that objects that appear and disappear in the sensor range of particular RFID tag readers.

- Perception of high-level events with low volume data. The special-purpose sensors generate very low sensor data volume, and generate sensor events highly correlated with robot tasks, such as activity recognition. For example, the ubiquitous robotics system can recognize that people have breakfast by cups and plates disappearing from the cupboard, appearing shortly after on the table, and finally moving into the dishwasher [18].

- Understanding everyday activity. The sensors in the network enable ubiquitous robots to observe activities very reliably, and comprehensively, over extended periods of time. Activity observation can be at different levels of abstraction. The robot can recognize activities, by interpreting the appearance

0921-8890/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.robot.2008.06.010
and disappearance of objects at task-relevant places, or by segmenting continuous movements into discrete subtasks. Detecting force-dynamic events, such as picking up an object and putting it down, allows segmentation of manipulation tasks into reaching, lifting, transfer, and placing subtasks. Finally, multiple cameras observing kitchen activity from different viewpoints, enables us to accurately track human body poses. Taken together, the sensor network can provide a comprehensive perception of kitchen activity that can be used for passively learning informative activity models.

- **Object recognition.** Object recognition and localization can be greatly simplified, by tagging task-relevant objects with unique identifiers, and by sensing these identifiers with RFID tag readers.

- **Increased observability of the environment.** Because the sensors in the network are spatially distributed, the robot can use them to compensate for the limitations of its onboard sensor equipment. The sensor network will thereby enable the robot to better perform joint household activities, such as setting the table, together with a human.

However, in order to enable effective ubiquitous robotics research, robot control systems need additional capabilities, in particular, at the middleware programming level. Compared to deploying a single autonomous robot with a homogeneous software infrastructure, ubiquitous robotics deals with orders of magnitude larger sets of sensors and effectors. These sensors and effectors are to be discovered at execution time, and thereafter to be used as resources of the robot.

This requires that there is not only data exchange, but also that a robot must infer the meaning of the data broadcast by a sensor, in order to use this sensor as a resource. For example, that an RFID tag reader reports the identification tags 124, 98, and 178 does not tell the robot anything. Only after the robot has discovered that the respective RFID tag reader is mounted in a particular cabinet, can it use the sensor to determine the objects that are in this particular cabinet. Other challenges of ubiquitous robotics, include the heterogeneity of the hardware and software infrastructure, the need for synchronization and sensor data fusion, and the required uptime and reliability of the sensors and effectors in the network.

We have developed an integrated solution to these problems, in the form of a service robot that can operate intelligently in an instrumented kitchen. In this paper, we report on our experience with this system, focusing on the following contributions:

1. **Design.** We propose a coherent modular design for a service robot, following the ubiquitous robotics paradigm. This design encapsulates and hides incompatible native software interfaces, and integrates them easily and efficiently into one decentralized Network Robotic System, with additional logging and synchronization mechanisms, and mechanisms for querying multiple sensors, and combining the resulting data. We explicitly support the discovery of networked sensing devices, and the management of sensor interface descriptions (their activation and the broadcast structure of the sensor data).

2. **Implementation.** We have implemented our design as a library of interfaces and drivers that support a variety of sensing, actuation, and computing devices. We include active sensing mechanisms, that allow robots to infer what information is provided by particular sensors, such as routines for localizing sensors and determining their operating range. Because a Network Robotic System offers redundant information, the active perception module deals with making sense of the data, exploiting its salient components, and minimizing communication overhead.

Our implementation extends the popular Player project, which develops open source software for robot control, and simulation [28]. Our implementation is also open source, and we encourage others in the research community to use and improve it.

3. **Usage.** We present illustrative usage examples from our experience in testing the implemented system. These examples demonstrate the power of our design, the flexibility of our implementation, and the complexity of tasks that it can handle. To our knowledge, this work represents one of the most capable and sophisticated service robot systems demonstrated to date.

The novel aspects of the above contributions include: new interfacing techniques for ubiquitous devices detailed in Section 6.1; integration of point cloud models, which is the basic building block for enabling object-based representations of the world, and their usage in robot manipulation detailed in Section 6.2; and the incorporation of mechanisms for active perception and sensor discovery, that are discussed in Section 6.3, and further explained in Section 7.

In the remainder of the paper we proceed as follows. Section 2 presents related work, followed by a description of our application domain in Section 3. We present our design requirements for operating in this domain in Section 4. Section 5 gives relevant background on the Player architecture, and in Section 6, we present the design and implementation of our system. We describe illustrative usage examples in Section 7, and conclude with Section 8.

## 2. Related work

In our work, we draw heavily on decades of research in operating systems and networks communities, for they have faced the same device-interface issues that are central to robotics research. For example, we employ the “device-as-file” model, which originated in Multics [6] and was popularized by UNIX [20]. We use well-understood networks techniques, including platform-independent data representations [17] and layered protocol specifications [25]. For an extensive survey of robot middleware packages, including Player, see [10].

Similar initiatives have been presented in [24,16,5,4], just to name a few. In [16], RT-Middleware, a system for controlling a robot arm, and a life-supporting robot system has been developed. The system is based on the well-known ACE (Adaptive Communication Environment) and CORBA (Common Object Request Broker Architecture) communication software infrastructures, thus making it easy to deploy it on any platform supported by them. RUNES (Reconfigurable Ubiquitous Networked Embedded Systems) [4] is a consortium project between several partners, both from academia and industry, with the purpose of creating large-scale, heterogeneous network-embedded systems, that interoperate and adapt to their environments. While the project targets a wide range of application domains, and not just mobile distributed robotics in particular, in-home healthcare through sensor monitoring seems to be the amongst the supported scenarios. A Java-based system for networked sensor environments is presented as part of LIME (Linda in a Mobile Environment) and its family of extensions (TinyLIME and TeenyLIME) [19,5]. LIME is based on the Linda model, where processes communicate through a shared tuple space, that acts as a repository of data tuples representing information. The concept of a PEIS (Physically Embedded Intelligent Systems) Ecology, which connects together standard robots, with simple off-the-shelf embedded devices, is

---

1. [http://playerstage.sourceforge.net](http://playerstage.sourceforge.net)
presented in [24, 2]. While the above mentioned initiatives have their strengths and weaknesses, none of them was able to fully satisfy our expected design requirements.

If we look at complex autonomous systems, it is typical to find that their components are usually written in more than one programming language. For example, real-time autonomous robot control architectures usually make use of the C++ programming language, while many AI planning approaches require LISP. Furthermore, several knowledge representation systems modules or machine learning frameworks are already implemented in various different languages.

We think it is unreasonable for an infrastructure to require the system components to be implemented in a specific language, thus preventing these systems for being deployed in complex scenarios.

The Player project already has comprehensive physically realistic simulation tools, that simulate sensing as well as robot actions and their effects. The availability of such comprehensive simulation infrastructure is necessary for the development of autonomous control systems that perform long-life learning.

Work related to other aspects of our project is covered in [11, 12, 22].

An important characteristic for a robot middleware, is that it must keep up with the rapid advancements in the field, and thus be as flexible as possible. Since its inception, the Player project has been consistently and continuously evolving. Dozens of research laboratories and institutions around the world are currently involved in the active development of Player, with even more simply using the software. Overall, we believe that one of the key features of a project of this size, is ease of maintenance (achieved in Player’s case through simplicity and transparency of the developer API), because the pool of researchers and developers can change rapidly [3].

3. Scenario

In ubiquitous robotics, a typical setting is the following one. A service robot establishes a connection to, and makes itself part of the ubiquitous computing, sensing, and actuation infrastructure. Having established the connection, the robot then perceives what is inside a cupboard in the same way as it perceives what is in its hand: by simply retrieving the respective sensor data and interpreting it — although it may not be physically connected to the sensor in question.

Our running example will be a mobile service robot (Fig. 1) that is tasked to set a table. To avoid the complexity of directly programming the robot to execute this task, the robot acquires the skills for setting a table through imitation learning, where our sensor-equipped kitchen, the AwareKitchen, observes people acting in the environment. The robot learns activity models from these observations, and uses the acquired action models as resources to learn high-performance action routines. This is an interesting and challenging problem, because it involves complex manipulation tasks, the acquisition and use of 3D object maps, the learning of complex action models, and high-performance action routines, and the integration of a ubiquitous sensing infrastructure into robotic control — aspects that are beyond the scope of current autonomous robot control systems.

Let us consider the deployment of an autonomous service robot in a new environment. The robot has various models of itself, including CAD and appearance models, that can be used to infer that a particular sensor might have the robot in its view. The robot also knows about possible sensing tasks that can be performed with certain sensors. For example, the robot knows that magnetic sensors can be used to recognize whether containers are open or closed, or that RFID readers can provide information about objects of interest in a certain area. To keep matters simple, we restrict ourselves to task settings where the robot is, in the installation phase, the only one acting in the environment.

Our autonomous service robot is built upon a RWI B21 base, equipped with a stereo camera system and laser rangefinders, as its primary sensors. To facilitate manipulation capabilities, two six-degree-of-freedom arms with simple grippers have been added to the base configuration [22]. Each arm features a smaller laser sensor, and an RFID reader for object identification.

The sensor-equipped kitchen environment is presented in Figs. 1 and 2. It consists of RFID tag readers placed in the cupboards, for sensing the identities of the objects placed there. The cupboards

---

2 http://playerstage.sourceforge.net/wiki/PlayerLsers.

also have contact sensors that sense whether the cupboard is open or closed. A variety of wireless sensor nodes equipped with accelerometers and ball motion sensors are placed on objects, and other items in the environment. Light and temperature sensors, together with other wireless sensor network nodes, have been scattered throughout the room in strategic places. Several small, non-intrusive laser range sensors were placed in the environment to track the motions of the people acting there (see Fig. 13).

The kitchen table is equipped with a suite of capacitive sensors, that essentially report the capacitance of different areas on the table when an object is placed there, as well as four RFID readers. In addition, seven cameras are mounted such that they cover the whole environment. Finally, machines and tools in the kitchen are also equipped with sensors [11].

Small ubiquitous devices offer the possibility to instrument people acting in the environment with additional sensors, and use the output as training data for machine learning applications. In our case, we have built a glove equipped with an RFID tag reader (Fig. 6), that enables us to identify the objects that are manipulated by the person who wears it. In addition, the person is equipped with small inertial measurement units (XSens MTx) that provide us with detailed information about the person's limb motions (Fig. 13).

4. Design requirements

To support application scenarios, such as the kitchen-service robot described in the previous section, we require infrastructure with the following capabilities:

Interoperability. The infrastructure must define interfaces and protocols that enable and facilitate communication between heterogeneous sensors and actuators. These protocols must exhibit an appropriate level of abstraction to support the development of portable algorithms for data fusion, perceptual processing, control, and higher-level reasoning. For example, it should be possible, with minimal programming effort, to combine an off-the-shelf robot (motors, encoders and controller), with a range sensor (sonar, laser, etc.) to form a mobile platform with basic navigation competency. Global location awareness is added to the robot by simply plugging in an existing, reliable localization module and supplying a map. Should the robot need to operate in an unknown or changing environment, a mapping module is added. When a new mapping algorithm has been developed, it can be dropped in as a direct replacement for the previous one. The key to this level of interoperability is the abstraction of the internal details of sensory, motor, and computational modules into well-defined interfaces.

Flexibility. From an architectural point of view, the system must be as flexible and as powerful as possible. For infrastructure to become widely adopted by the community, it must impose few, if any, constraints on how the system can be used. Specifically, we require independence with respect to programming language, control paradigm, computing platform, sensor/actuator hardware, and location within a network. In other words, a researcher should be able to: write a control program in any programming language, structure the program in the best way for the application at hand, run the program on any computer (especially low-power embedded systems), make no changes to the program after integrating new hardware, and remotely access the program over a network. Though not a strict requirement, we also aim to maximize the modularity of our architecture, so that researchers can pick and choose the specific components that they find useful, without using the entire system.

Simulation. We require a realistic, sensor-based simulation. Simulation is a key capability for ubiquitous computing infrastructure. The main benefits to the user of simulation over real hardware, are convenience and cost: simulated devices are easier to use (e.g., their batteries do not run out), and much cheaper to acquire and maintain. In addition, simulation allows the researcher to explore system configurations and scales that are not physically realizable, because the necessary hardware is not available. The simulation must present the user with the same interface as the real devices, so that moving an experiment between simulation and hardware is seamless, requiring no changes to the code.

Though they may seem lofty, these goals are in fact achievable, as we explain next.

5. Background: The Player architecture

Because it fulfills our design requirements, we extensively maintain, use, develop, and extend the open source Player software suite, which is freely available for download. The Player project (formerly known as Player/Stage) produces tools for rapid development of robot control code [9]. The project's three primary tools are: Player, Stage, and Gazebo. Player is a hardware abstraction layer for robotic devices [27]. Stage and Gazebo are, respectively, 2D and 3D multiple-robot simulators (Fig. 3).

The goal of the Player project is to produce communal robot and sensor network infrastructure, that improves research practice and accelerates development, by handling common tasks and providing a standard development platform. By collaborating on a common system, we share the engineering burden and create a means for objectively evaluating published work. If you and I use a common development platform, then you can send me your code and I can replicate your experiments in my lab.

The core of the project is Player itself, which functions as the OS for a sensor-actuator system, providing an abstraction layer that decouples the user’s program from the details of specific hardware (Fig. 4).
Player specifies a set of interfaces, each of which defines the syntax and semantics for the allowable interactions with a particular class of sensor or actuator. Common interfaces include laser and ptz, which respectively provide access to scanning laser range-finders and pan-tilt-zoom cameras. A hardware-specific driver does the work of directly controlling a device and mapping its capabilities onto the corresponding interface. Just as a program that uses an OS’s standard mouse interface will work with any mouse, a program that uses Player’s standard laser interface will work with any laser — be it a SICK or a Hokuyo range-finder. These abstractions enable the programmer to use devices with similar functionality identically, thus increasing the transferability of the code [28].

The 2D simulator Stage and the 3D simulator Gazebo (Fig. 3) also use a set of drivers to map their simulated devices onto the standard Player interfaces. Thus the interface that is presented to the user remains unchanged from simulation to hardware, and back. For example, a program that drives a simulated laser-equipped robot in Gazebo, will also drive a real laser-equipped robot, with no changes to the control code. Programs are often developed and debugged first in simulation, then transitioned to hardware for deployment.

In addition to providing access to (physical or simulated) hardware, Player drivers can implement sophisticated algorithms that use other drivers as sources and sinks for data (see Fig. 5). For example, the lasercspace driver reads range data from a laser device, and convolves that data with the shape of a robot’s body to produce the configuration-space boundary [13]. The lasercspace driver’s output conforms to the laser interface, which makes it easy to use. Other examples of algorithm drivers include adaptive Monte Carlo localization [7]; laser-stabilized odometry [14]; and Vector Field Histogram navigation [26]. By incorporating well-understood algorithms into our infrastructure, we eliminate the need for users to individually re-implement them.

Network access to devices is provided by way of a client/server transport layer, that features auto-discovery mechanisms. The details of interacting with the transport layer are encapsulated in client libraries, that simplify the development of custom applications. Because of the standards that Player imposes on its architectural and transport layer, client libraries are available for a variety of programming languages, including C, C++, Java, Lisp, Python, Ada, Tcl, Ruby, Octave, Matlab, and Scheme.

One can think of the Player driver system as a graph (Fig. 12), where nodes represent the drivers, which interact via well-defined interfaces (edges). Because it is embodied, this graph is grounded in the robot’s (physical or simulated) devices. That is, certain leaf drivers of the graph are connected to sensors and actuators. Internal drivers implement algorithms (e.g., localization), and are connected only to other drivers. Because the interfaces are well-defined, drivers are separable in that one can be written without knowledge of the internal workings of another. If my algorithm requires data from a laser range-finder, then the driver that implements my algorithm will take input over a laser edge from another driver; I do not care how that data gets produced, just that it is standard laser data. A wide variety of control systems can be constructed by appropriately configuring and connecting drivers. The control system is also accessible from the outside; an external program (e.g., a client) can connect to any driver, via the same standard interfaces. The system is, to a limited extent, reconfigurable at run-time, in that control programs can connect to and disconnect from devices at will.

Full run-time reconfigurability, by which an external program can...
change existing connections among devices, is a topic for future work.

6. Approach

We now describe our system design and its implementation on our kitchen service robot. Our implementation is freely available under an open source license.4

4 Some parts of our implementation are already available in the Player CVS repository on SourceForge, and the rest will be made available soon, together with the release of Player 3.0.

6.1. Interfacing ubiquitous sensing devices

Building a common infrastructure for heterogeneous devices such as Wireless Sensor Networks, RFID technologies and Inertial Measurement Units, is motivated by the fact that major manufacturers use their own protocols, without consideration of interoperability issues between their products, and products from other companies. Therefore, the sensors they produce have incompatible interfaces, and sometimes the interfaces are even buried in single specific applications that the sensors are made for. This situation seems unlikely to change any time soon.

Part of our efforts have gone towards supporting a series of new hardware platforms, and making them available to the community,
including:

- **Wireless Sensor Networks**— with a wide variety of different sensor nodes, ranging from the RCores and Particles from TeCo/Particle Computers to the Mica2 and Mica2Dots from Crossbow, as well as customized sensors such as the Porcupines\(^5\);
- **RFID technologies**— several readers such as the Inside M/R300, SICK RFI 341, the Skyetek M1/M1-mini;
- **Inertial Measurement Units**— popular devices like the XSens MT9/MTx, which provide drift-free 3D orientation and kinematic data.

An overview of how these different technologies are connected, and work seamlessly one with each other is presented in Fig. 6. The information exchange between sensor nodes is supported via PSFNs (Player Sensor Fusion Nodes). A Player Sensor Fusion Node, is a wirelessly enabled device (e.g., a Gumstix embedded computer) that has the capability to run a POSIX-like operating system and Player, thus providing access on the network to some of the ubiquitous sensing devices. The number of PSFNs, however, does not depend on the number of the other sensing devices, as a PSFN can also provide support for filtering data or doing any type of in-network data processing.

One example of a useful PSFN is to permit the acquisition, interpretation and usage of data between incompatible devices, like the TeCo Particles and the Mica2 motes. Another example would be to read data from a 2D laser sensor, and whenever an obstacle enters in its field, a PSFN node could trigger an alarm by sending a “piezo enable” command to all buzzer-equipped Mica2 motes.

The wireless sensor nodes are connected together in a mesh network and can exchange information among themselves, as needed by the user application. Some of the nodes are able to run complex applications from the TinyOS repository, while others have much simpler purposes. For example, a ball-motion wireless sensor will only send a digital value when the state of the device has changed (e.g., a movement took place), while another node can provide support for both collecting measurements from adjacent nodes, and for routing packets in the network.

Our analysis of what ubiquitous sensing devices could provide in terms of sensor data, led to the creation of the **wsn**, **imu**, and **rfid** interfaces and the associated data structures shown in Fig. 7. The data packets are easily extensible, and the recent addition of a mechanism in Player that allows reconfigurable interfaces simplifies things further.

Besides providing access to hardware devices, a number of drivers that take care of sensor calibration, data fusion, synchronization and logging, were also implemented. As show in Fig. 6, the preferred location for these drivers is on a fusion node. An example is the acceleration calibration driver, *accel_calib*, which receives acceleration data from a **wsn** interface (eg. **wsn**:0), computes the calibrated values and sends them along the pipeline via another **wsn** interface (eg. **wsn**:1). A fusion node is also responsible for storing log files of the sensor data locally.

Automatic feature extraction drivers were also developed [12, 22]. The *accelfeatures* driver takes care of extracting relevant features from acceleration data, such as: the mean and standard deviation of the signal, the magnitude, skewness, kurtosis, RMS and energy. It also performs decomposition and analysis, using Fourier coefficients, wavelets, as well as ICA (Independent Component Analysis) or PCA (Principal Component Analysis). By starting multiple instances of the driver, the output of one instance can be used as the input of another. Fig. 8 depicts the usage of our automatic acceleration feature extraction driver, for nodes in a Wireless Sensor Network. Note that one instance of the driver runs on one Player Sensor Fusion Node (here depicted with IP address 10.0.0.1), while the second instance runs on another node (with IP address 10.0.0.2), thus showing the simplicity of distributed processing in a Player network.

In this case, the *accelfeatures* driver will be started on two different Player nodes (10.0.0.1 and 10.0.0.2), using the output of the first as the input of the second. Therefore, the first instance will receive acceleration data via the **wsn**:0 interface, calculate wavelet coefficients using Daubechies 20 and perform Independent Component Analysis (ICA) in parallel, and finally pack the resulting values in the **wsn**:1 interface. The second instance will take the results from **wsn**:1 as input, and will compute standard features such as energy, RMS, magnitude, and so on, and will provide the results to the user via the **wsn**:2 interface. The calculated features are used for learning models of human motions from acceleration data (Section 7).

From the user’s point of view, the entire architecture can be easily accessed from any of the supported programming languages. In most cases, the user needs to first create a connection with the Player server, and then access the information, using one of the predefined interfaces, thus abstracting any of the lower level internals of the hardware connections. An example of how this can be done for the **wsn** interface, using the Player Java client is shown in Fig. 9.

Besides driver development, the integration of ubiquitous sensing and computing infrastructure yields interesting challenges, like the incorporation of new sensing infrastructure during operation.

---

Fig. 8. Usage example for the `accelfeatures` driver.

```java
// Player configuration file for PSFN 1 (10.0.0.1)

driver {
  name "accelfeatures"
  provides ["features:" "10.0.0.1::wsn:1"]
  requires ["wsn:0"]
  window_size 16
  queue_size 10000
  overlapping 50
  feature_list ["wavelet.coefl" "ica"]
  wavelet_params ["daubechies:20"]
}
```

Fig. 9. WSN interfacing example using the Java client.

```java
Player JavaClient Application snippet for a Wireless Sensor Network:

WSNInterface wsn = client.requestInterfaceWSN (index, accesscode);
PlayerWSNData data = packet = wsn.getData ();
DATA packet

wsn.sendCommand (PlayerWSNCommand.setLed (green, on));
COMMAND packet

wsn.requestFrequencyRate (PlayerWSNCommand.set (new_frequency);
reply = wsn.readReply ();
REQUEST/REPLY packet
```

and the active querying of sensor networks for specific information, such as happens in TinyDB [15].

Fig. 10 shows the lower level connections with the distributed hardware infrastructure installed in our laboratory. Each node, be it a personal computer or a small embedded device like the Gumstix, is running a Player server, and provides access to some hardware devices. The sum of these Player-enabled nodes forms what we call the level 1 layer of our distributed system.

The level 2 layer comprises all the Player Sensor Fusion Nodes that run filters, data fusion, and processing algorithms, and either use the level 1 connections to get data from a sensor, or control an actuator, or act as data producers and consumers themselves. Examples of such devices are shown in Figs. 6 and 12.

We define a layer 1 node as a Player-enabled node (it can be a personal computer, a laptop, a PDA, or any other small hardware device that can run a POSIX-like OS) that provides access only to hardware devices (sensors or actuators). An example of hardware devices are the Mica2 motes, which are connected to the system via a level 1 Player node (running the `mica2` Player driver), through an MIB510 serial connection.

In contrast, a level 2 node acts like a “higher-level” Player-enabled node, in the sense that it is configured to run data filters, probabilistic localization modules, navigation algorithms, model learning modules, and so on. A level 2 node receives data from and controls a level 1 node. For example, a localization module such as AMCL (Adaptive Monte Carlo Localization) needs access to a sensor such as an ultrasonic array, or a 2D laser, and to an actuator such as a mobile robotic base, all provided by level 1 nodes, in order to execute its particle filtering algorithm. A simpler example of a level 2 node running a feature extraction algorithm, is the `accelfeatures` driver (Fig. 8).

From an architectural point of view, a mobile robotic platform, such as the one comprising nodes 7 and 8 in Fig. 10, is not distinct from the other resources in the distributed robotic network: it just provides access to a set of sensors and actuators.

### 6.2. Supporting model learning

Few existing robot systems make use of context history. Moving towards systems that automatically build models from their past experience, and use them together with online information, is an imperative shift that must be done, if we seek to deploy them in complex scenarios. A context-aware system can be of much greater help, and will interact in a more efficient and friendly manner with its users. By analyzing previous usage information, and predicting its current and future state, it might be able to recognize and understand users’ actions.

We have implemented several modules that support model learning, and open the door towards high level cognitive processing tasks. Our approach is twofold: (i) we acquire 3D environmental models and maps, because we aim to support intelligent robotic assistants that can roam around and manipulate objects; and (ii) we acquire libraries of models for human-like motions, which are then used as blueprints for identifying, classifying and reconstructing complex movements of people.

To perform robotic manipulation in a complex environment, the system needs a fine-grained 3D polygonal map, updated as often as possible. In our work such maps can be acquired, using a probabilistic algorithm. The algorithm gets an unorganized 3D point cloud as input, and provides a polygonal description containing higher level semantics as output. An overview of the algorithm is presented in Fig. 11. As the mathematical explanation
of the underlying algorithms falls outside the scope of this paper, the reader is encouraged to consult [21,22].

Model acquisition based on planar surfaces is well-suited for mapping indoor environments. Fig. 14 depicts results of the environmental model learning process.

In order to support the creation of such maps, interfaces and drivers that support 3D point cloud data acquisition and processing, have been developed. Point clouds can be acquired either from hardware devices that are capable of sending 3D point coordinates as output, or from drivers that implement data fusion algorithms that combine sensory data from several hardware devices. We support a variety of hardware devices, including: the Swiss Ranger SR3000 (a time-of-flight range camera, that supports the acquisition of 3D point clouds), the SICK LMS5400 (a highly accurate 2D laser measurement system), and Leutron Vision’s PicPort Stereo (a framegrabber that supports the acquisition of 3D point clouds using stereo vision), to name a few.

Additionally, we have developed a large pool of devices that support the acquisition or processing of point clouds, such as filters for laser data (e.g., lasercutter removes “unwanted” rays), data fusion (e.g., laserptzcloud, laseractarraycloud, and laserlimbcloud fuse position readings from the ptz, actarray or limb interfaces together with laser data, perform coordinate transformations and return the results as 3D point clouds) or kinematics (e.g., eeDHcontroller computes the necessary joint commands for an array of actuators using Inverse Kinematics routines in order for the end-effector to reach a certain position in space; while moving the arm towards the goal, the positions of the end-effector are computed, using Forward Kinematics and returned to the user).

One usage example of the above mentioned devices is demonstrated in Fig. 12. The devices depicted on the left side of the picture connect to hardware sensors (sr3000, sicklms400, urglaser and picportstereo) and actuators (amtecM5 and ptu64). All the other devices are virtual, in the sense that they act as data consumers and producers. As explained before, for example, the eeDHcontroller driver does Inverse Kinematics and Forward Kinematics calculus (based on the Denavit–Hartenberg parameters) for the amtecM5 device using the actarray interface. It first computes the necessary joint commands for a given end-effector pose, then it commands the actuators until the goal pose is reached, providing up-to-date information on the end-effector’s pose on the way. Given a 2D laser ranger placed on the end-effector, its pose is used by the laserlimbcloud driver together with the data coming from the laser sensor, and returned as a 3D point cloud.
6.3. Active perception and sensor discovery

Active Perception addresses the problem of what sensors to use, and where to place them in a ubiquitous sensor-equipped environment [8]. It is characteristic for sensor-equipped environments to have sensors that provide redundant information, thus evidencing that some state variables are provided by multiple sensors. In particular, given $n$ sensors that provide overlapping information, we are interested in maximizing the probability $P$ that the system is in a state $x$, by minimizing a cost function (e.g., the number of sensors used or restricting the classes of sensors to be passive, or non-intrusive).

Another aspect is that the sensors only provide partial information about the state we are interested in. For example, an RFID tag reader in the glove senses that an object with a certain ID...
is close to the glove, but it cannot tell us whether the object is in the hand, the state we might be interested in.

Our approach combines the known Active Perception concept with Sensor Discovery, in the sense that, while we are interested to place sensors in the environment so that they give the best information possible, we would also like our system to automatically configure itself to some extent, by identifying the sensor streams, and assigning the proper labels to them.

The placement of sensors is very important in a distributed robotic system, and special care has to be taken when configuring the network. By investigating and automatically identifying the sensor models (poses, shapes, and usage) in a distributed network, we can maximize the information efficiency of the system, which conceptually can lead to the idea of a self-calibrating sensor network.

For example, consider a mobile robot navigating through an indoor environment. Upon arriving in the new environment, the robot connects to the existing sensor network of the sensor-equipped kitchen. In the beginning it receives the sensor data, but it has no idea of what the data means. In order to use the sensors of the kitchen as its resources, the robot explores the environment, inferring which information the sensors provide. For example, the robot uses previously acquired models of itself to decide whether the sensors detect it. In many cases, having the model of the perceived object, knowing its own state, and interpreting the received sensor data, the robot can infer the position and orientation of sensors. These concepts have been integrated into our design, and several results are presented in Section 7.

Self-calibrating sensor networks could also be useful in a learning phase, where the environment is equipped with additional sensors that are used to provide “ground truth” information, as needed by the supervised learning algorithms.

7. Usage examples

We tested the ubiquitous sensing infrastructure in several experiments, lasting tens of hours. The complete sensor network was controlled by our architecture at all times, and acted as one decentralized network robotic system. Our experiments comprised data analysis and model learning applications on several large sensor data collections, gathered during extended periods of time [1,2,11].

Several persons have been instrumented with tiny inertial measurement units on their limbs, as well as with an RFID-enabled glove, and their movement observed while they were setting up the table. The data was later used to segment and reconstruct their motion in space (Fig. 13), as well as learn models of ADLs (Activities of Daily Living). Having all the sensor data synchronized, and logged from several fusion nodes was a critical requirement for the system.

By emphasizing the interface abstractions, we were able to collect data, without bothering at all about the hardware details of the sensors and actuators installed. For example, acceleration data from different sensor nodes was gathered without worrying whether the underlying architecture of the node was based on TinyOS (Mica2/Mica2Dot) or our own (Particles/RCore). Results regarding activity recognition in the kitchen environment [11,1] fall outside the scope of this paper.

We also used our distributed Player network to collect 3D point clouds from various devices, and used them to perform environmental model learning. Since we are dealing with a household robotic assistant, we want to make sure that the robot learns and understands the environment it operates in, not only in terms of distances and points, but in higher-level semantics. The process of acquiring such semantic maps has been described in [21], and our results are shown in Fig. 14.

Looking through the eyes of household robot, the kitchen is a room that consists of a set of cabinets, which are containers with front doors, shelves, which are open containers, and tables, which are horizontal planes that can be used for performing kitchen work, having meals, etc. There might be more pieces of furniture that have important roles for the robots’ tasks, but for now we only consider these. All other types of furniture, such as chairs, are uniformly considered to be obstacles. Cabinets are cuboid containers with a front door that might contain objects of interest. Some cabinets have specific purposes such as the oven, the refrigerator, and the dishwasher. The other cabinets are used to store objects. Tables are another important subclass of furniture-like objects. They are horizontal rectangular flat surfaces approximately at hip height, that satisfy some given minimal size and length requirements. Similarly, we have drawers and shelves as additional furniture-like object classes.

The raw data is taken from the level 1 Player devices and it is passed through a number of processing steps, such as coordinate transformations, geometrical features calculus, registration, and finally segmentation. By applying some of the above mentioned assumptions, we infer these areas of interest in which the robotic assistant could operate, thus giving a whole new meaning to the map (Fig. 14).

The flexibility offered by our system in terms of dealing with decentralized virtual devices that can use each other’s inputs and outputs, as well as the possibility to replay mapping experiments from logged data, is crucial while dealing with 3D object maps.

We demonstrate the usage and importance of having a detailed semantic map of the environment in an application scenario, using our mobile robot. Given a robot equipped with manipulation capabilities in a household kitchen environment, our goal is to learn what a cupboard is, where it is located, and then to open it. Obviously, in order to interact with the real world and perform such complex manipulation tasks, the robot needs a detailed 3D description of the environment, such as what type of cupboard and handle it has to deal with.

After building such a map, the robot can proceed to manipulate the environment. For our 6-DOF arm, we describe the arm’s trajectory as a curve in space, using the following algebraic formula: 
\[ ax^2 + bxy + cy^2 + dx + ey + f = 0. \]
Our goal is to follow this curve by discretization into small steps and reach every point. This is done by computing the joint vector \( q = (\theta_1, \ldots, \theta_n) \) for each matrix obtained for a point \( p \) on the curve.

Fig. 15 shows the trajectories in space of all the arm’s joints for opening a cupboard door, whose position and handle were detected, and labeled accordingly in the semantic map. Once the cupboard door has been opened, the associated magnetic sensor installed on the door (Fig. 1) will change its status, and therefore, the sensor will be identified and the appropriate label will be given to the cupboard, thus performing sensor discovery through manipulation.

A simple laser sensor installed in the kitchen is enough for determining the relative position of the robot, and of the sensor itself. By determining the position of the laser in the environment, and assigning an entry in the semantic map, the robot will be able to use that sensor in the future as part of its distributed sensing capabilities. In our example, we assume that the mobile robot is localized with respect to the environment, and we try to detect the absolute position in the room of the laser sensor. Since our mobile robot is cylindrical in nature, we assume that looking for arcs in the set of laser range measurements suffices [30]. During the time the robot moves in the room the laser is queried repeatedly, and arcs are determined. By assuming either that the robot is the only object moving in the scans, or that the robot knows its own geometric structure, we can determine that the feature that constantly changes its position, is, in fact, the robot (Fig. 15).
Fig. 15. Top left: robot arm trajectory while opening a drawer. Bottom left: the robot opening a drawer. Top right: the robot’s movement in the kitchen as seen by a laser sensor installed in the environment; by assuming that the robot geometry is known or that the robot is the only entity moving in the environment we can detect the laser’s position on the wall, thus making it a valid sensor for the robot’s usage in future. Bottom right: the robot identifies objects in the drawer by the use of RFID tags, using its onboard RFID reader.

Fig. 16. From reality to simulation (Gazebo) and back: The AwareKitchen.

Similar to the results presented in Fig. 15, we swapped the roles of the RFID tags and readers, by installing RFID readers in the drawers and placing an RFID tag on the end-effector, and successfully identified it and assigned semantic entries in the map. Before attempting to use our robot assistant in the real world, we go through several stages of fine-grained simulation of our plans and controllers. To perform an accurate 3D simulation, we make use of the Gazebo simulator, under which we created an environment similar to the real one (Fig. 16). By introducing error models in our sensor data, and employing probabilistic techniques in the simulator, we can achieve similar results in the real world, without changing our programming infrastructure or algorithms.

8. Summary and future work

In this paper we have reported on our experience in developing an intelligent service robot, that is designed to automate common tasks in a kitchen setting. Whereas most in the research community pursue the monolithic approach, in which all sensing, actuation, and computation is performed on board the robot, we follow the ubiquitous robotics paradigm, in which the robot is but one part of a network of sensors, actuators, and computers. By exploiting these other devices, a robot can perform more complex tasks without becoming overly complex itself.

Our work is guided by a coherent modular design, that encapsulates incompatible native software interfaces and integrates them easily and efficiently into one decentralized Network Robotic System. We have implemented this design as a library of interfaces and drivers, that support a variety of sensing, actuation, and computing devices. We have extensively tested our implementation in a variety of usage scenarios, demonstrating the power and flexibility of our approach, as well as the complexity of tasks that it can handle.

We advocate the development and use of open source infrastructures for accelerating research in this field. Substantial parts of the work described in this paper, including devices offering hardware connectivity, logging, synchronization, as well as virtual devices for feature extraction and data fusion, are already included in the open source Player project. Higher-level services, in particular environmental model learning, and automatic acquisition of activity models are still under investigation and development, and will be contributed to the Player project.

We believe that our work, as one of the most capable and sophisticated service robot systems demonstrated to date, forms a basis for further investigating the area of ubiquitous robotics. We have shown our approach to reliably support the development and deployment of leading-edge service robots, that can serve as robot assistants with comprehensive manipulation capabilities, acting in sensor-equipped environments.

Acknowledgments

This work has been jointly conducted by the Intelligent Autonomous Systems group at the Technische Universität München and the Artificial Intelligence Center at SRI, International. This work is supported by the CoTeSys (“Cognition for Technical Systems”) cluster of excellence and in part by the National Science Foundation under grant IIS-0638794. Special thanks go to the entire Player community for their continuous effort in developing and maintaining open source tools for robot and sensor applications.
References


Radu Bogdan Rusu is a Ph.D. student/researcher at the Intelligent Autonomous Systems group, Computer Science Department IX, Technische Universitaet Muenchen. He received his M.S. (2000) and M.Phil (2002) degrees from Technical University of Cluj-Napoca, Romania. He is currently affiliated with the SRI Artificial Intelligence Center as an International Visiting Researcher, working in the area of perception and 3D semantic mapping. During the last couple of years, he served as a developer and contributor for the Player project, as well as maintainer of the open source JavaVinci library. Radu Bogdan Rusu was one of the main organizers of the first Player Summer School for Cognitive Robotics (PSSCR07), which brought together over 50 international students from all over the world. He also served as a reviewer for robotics conferences such as IROS or ICRA and is a member of IEEE Robotics and Automation Society.

Brian Gerkey is a research scientist at Willow Garage. He was previously a computer scientist at the SRI Artificial Intelligence Center and a postdoctoral researcher in the Stanford Artificial Intelligence Lab. He received his Ph.D. in Computer Science from the University of Southern California in 2003. Dr. Gerkey has worked with robotic systems for over ten years, from low-level controller design to high-level multi-robot coordination. In his current work, he focuses on techniques for improving manipulation for indoor robots and mobility for outdoor robots. Dr. Gerkey is co-founder and maintainer of the Player Project.

Michael Beetz is a professor for Computer Science at the Department of Informatics of the Technische Universitaet Muenchen and heads the Intelligent Autonomous Systems group. He is also managing director of the German cluster of excellence COTESYS (Cognition for Technical Systems) where he is also co-coordinator of the research area “Knowledge and Learning”. Michael Beetz received his diploma degree in Computer Science with distinction from the University of Kaiserslautern. He received his M.Sc, MPhil, and Ph.D. degrees from Yale University in 1993, 1994, and 1996 and his Venia Legendi from the University of Bonn in 2000. Michael Beetz was a member of the steering committee of the European network of excellence in AI planning (PLANET) and coordinating the research area “robot planning”. He was also principal investigator of a number of research projects in the area of AI-based robot control. His research interests include plan-based control of robotic agents, knowledge processing and representation for robots, integrated robot learning, and cognitive perception.