Master Thesis
Homes for Robots: A Rapid Prototyping Toolkit for Robotics and Intelligent Environments

Author: Triquell Conde, Oriol
Matriculation Number:
Address:

Advisor: Prof. Dr. Matthias Kranz
Begin: October 10, 2010
End: March 31, 2011
Abstract

Intelligent Environments (IEs) have much in common with Robotics. A great number of distributed, heterogeneous sensors, actuators, and processing components exchange data and communicate and thereby form an ecology of information processing systems. The complexity of the individual components and subsystems demands for a simulation environment that allows us to selectively investigate the characteristics and properties of systems, ideally already before we actually start building them. In the age of Ubiquitous Computing, sensing, actuation, communication and information processing is embedded in the world – in everyday objects, the environment and many other places. We therefore require a way to efficiently prototype and develop Intelligent Environments including their physical and digital parameters. This helps us improve the design or analysis of systems and applications, optimize their functionality, or to explore new ideas. In this Master Thesis, a rapid prototyping toolkit is to be developed that allows us to jointly simulate smart spaces, both for the domains of robotics and Ubiquitous Computing. The tool bases on an open source 3D CAD (computer aided design) software called Sweet Home 3D. This tool allows novice users to visually create rooms, floors and houses and explore their layout and appearance. The goal of this thesis is to develop means to allow us to easily export these environments to a robotics simulation software called ROS (robot operation system, a meta-operating system used in robotics) and its associated 3D simulation tool, Gazebo. This enables us to quickly prototype new environments and to model existing environments. This is currently a cumbersome and time consuming task. Sweet Home 3D is extensible and lets users add models of e.g. furniture, appliances, electronic devices, vehicles, plants, trees... any model in supported formats: OBJ, DAE, 3DS and LWS. These built–in and custom–made objects in a 3D file format need to be exported to URDF, the Unified Robot Description Format. It then can be used to create objects for the ROS/Gazebo simulation tool chain. In addition to simple objects, rooms need to be augmented with additional information, allowing to generate a connected graph to be used e.g. for human or robot navigation purposes. Finally, for standard parts of an environment, such as light, doors and windows, an addition to the exported data has to be created, allowing the control of these components directly via ROS controllers, e.g. to open a door or a window. For these application usage scenario, an example will be created to demonstrate the novel functionality of the developed system.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Introduction</td>
<td>5</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>5</td>
</tr>
<tr>
<td>1.2 Purpose</td>
<td>6</td>
</tr>
<tr>
<td>1.3 Thesis Overview</td>
<td>7</td>
</tr>
<tr>
<td>2 Background</td>
<td>8</td>
</tr>
<tr>
<td>2.1 Intelligent Environments</td>
<td>8</td>
</tr>
<tr>
<td>2.2 ROS - Robot Operating System</td>
<td>9</td>
</tr>
<tr>
<td>2.2.1 ROS concepts</td>
<td>9</td>
</tr>
<tr>
<td>2.3 Gazebo Simulator</td>
<td>11</td>
</tr>
<tr>
<td>2.4 URDF - Unified Robot Description Format</td>
<td>12</td>
</tr>
<tr>
<td>2.5 Sweet Home 3D - basis for a 3D CAD tool</td>
<td>13</td>
</tr>
<tr>
<td>3 Modeling and Simulation</td>
<td>17</td>
</tr>
<tr>
<td>3.1 Simulation in robotics</td>
<td>18</td>
</tr>
<tr>
<td>4 Designed application - Sweet Home 3D Extended</td>
<td>20</td>
</tr>
<tr>
<td>4.1 New Features</td>
<td>20</td>
</tr>
<tr>
<td>4.2 Limitations and problems</td>
<td>21</td>
</tr>
<tr>
<td>4.3 Requirements</td>
<td>22</td>
</tr>
<tr>
<td>5 Writing the application</td>
<td>23</td>
</tr>
<tr>
<td>5.1 Development Process</td>
<td>23</td>
</tr>
<tr>
<td>5.2 Modified Classes</td>
<td>24</td>
</tr>
<tr>
<td>5.3 New Classes</td>
<td>27</td>
</tr>
<tr>
<td>6 Conclusions</td>
<td>29</td>
</tr>
<tr>
<td>A HowTo - Tutorials</td>
<td>31</td>
</tr>
<tr>
<td>A.1 Import “Gazebo door or window”</td>
<td>31</td>
</tr>
<tr>
<td>A.2 Modify the revolute joint in gazebo doors or windows</td>
<td>32</td>
</tr>
<tr>
<td>A.3 Export the content to URDF and launch it in simulator Gazebo</td>
<td>32</td>
</tr>
<tr>
<td>A.4 Modify the exported files to improve the gazebo visualization</td>
<td>33</td>
</tr>
</tbody>
</table>
CONTENTS

B Sample 34
List of Figures 42
Bibliography 43
Chapter 1

Introduction

We are steadily moving into a new age of information and communication technology named as Ubiquitous Computing, idea envisioned by Weiser in 1991 [1], changing the everyday life of people. This research area has been recently named as Ambient Intelligence, it represents a new generation of user-centered computing environments aiming to find new ways to obtain a better integration of the information technology in everyday life devices and activities. Although its results were previously known as ”Smart Rooms” or ”Intelligent Environments”, in this thesis we will use this last term.

The vision behind this concept is the creation of rooms where humans interact in a natural and non-invasive way with computational services that help them in their everyday tasks. As unobtrusiveness and invisibility are goals of good Intelligent Environments, the user–interface primitives of these systems are not menus, mice and windows but gesture, speech, affect and context. So, the human computer interaction (HCI) paradigm we currently used on the desktop computers will not be sufficient [2]. We should be able to interact with all related computation devices as a whole, and express our intended tasks in a high abstraction level as we used to communicate with other people.

In the robotics domain, environments enriched with distributed sensor–actuator–systems, certain autonomy, and able to control their complex internal and external functions are considered as “Immobots”: Immobile Robots [3]. Therefore, Intelligent Environments can be considered as Immobots as well.

1.1 Motivation

Intelligent Environments are aimed at improving the inhabitants’ experience and task performance, acquiring and applying knowledge about you and your surroundings with a great number of distributed, heterogeneous sensors, actuators, and processing components exchange data and communicate and thereby form an ecology of information processing systems. As a result, our physical world is being pervasively digitally instrumented. This complex system uses a multi-disciplinary area of Context–Aware Computing, which
combines technology, computer systems, models and reasoning, social aspects, and user support. All of this increases the difficulty of researchers work.

Due to the complexity of the individual components and subsystems, development and implementation of ubiquitous computing applications require rapid prototyping [4] and early evaluation. However, the infrastructure to support such application is often not available at the time of application development, and therefore, it demands for a simulation environment that allows the researchers to selectively investigate the characteristics and properties of systems, ideally already before they actually start building them. We therefore require a way to efficiently prototype and develop Intelligent Environments including their physical and digital parameters. This helps us to improve the design or analysis of systems and applications, optimize their functionality, or to explore new ideas.

To model and simulate the environment, some specialized software is needed, this would enable us to quickly prototype new environments and to model existing environments. Is well known that the modeling task is currently a very cumbersome, time-consuming and costly task.

1.2 Purpose

The main goal of this thesis is to develop a rapid prototyping toolkit that enables users to jointly simulate smart spaces, both for the domains of robotics and Ubiquitous Computing. The application must allow users to visually create and design buildings with multiple floors, walls, rooms, appliances, furniture, doors, windows, lights... and helps to place all this objects on a 2D plan with a 3D preview. Then, it should be easy to allow us export the designed environment to a robotics simulation software called ROS (robot operation system, a meta–operating system used in robotics) and its associated 3D simulation tool, Gazebo.

The tool bases on an open source 3D CAD (computer aided design) software called Sweet Home 3D. Sweet Home 3D is extensible and lets users import models of any model in supported formats: OBJ, DAE, 3DS and LWS. But it needs to be extended allowing users easily add more floors to the building, allowing to augment the rooms information and generate a connected graph to be used e.g. for human or robot navigation purposes, and allowing to export the designed environment to URDF, the Unified Robot Description Format. It then can be used to create and simulate the objects for the ROS/Gazebo simulation tool chain. Finally, an addition to the exported data must be created, allowing the control of doors and windows directly via ROS controllers.

The functions implemented will have the source code open and fully accessible, released to the community in [https://vmi.lmt.ei.tum.de/ros](https://vmi.lmt.ei.tum.de/ros)
1. Chapter 1. Introduction: This chapter introduces the reader in the field of research of this thesis.

2. Chapter 2. Background: Explains a little bit the technical bases of this thesis.


4. Chapter 4. Designed application: In this chapter, the features, limitations and requirements of the designed application are described.

5. Chapter 5. Writing the application: The development process, the modified classes and the new classes to achieve the desired features of the application are explained here.

6. Chapter 6. Conclusions: A summary of the most important results of this thesis. Also possible future work is given.

7. Appendix A. How to – Tutorials: As his name suggest, in this chapter there are tutorials about how to use some features of the software.

8. Appendix B. Sample: An example of a building prototyped with the developed software.
Chapter 2

Background

Before we start discussing the developed software, it is convenient to know some background terms, concepts, and software.

2.1 Intelligent Environments

The term Intelligent Environment (IE) has been introduced in the previous chapter; however, a more detailed, but briefly, description is given below.

Intelligent Environments combine: Ubiquity, context-awareness \[5\], intelligence, and natural interaction \[6\]. They are adaptive systems that process multiple sources of input, interact naturally with the inhabitants in a ubiquitous manner, maintain awareness about the intentions of the inhabitants, and automate the environment. For that, they are equipped with sensors, devices, actuators, and computers that are networked with each other and the internet.

Characteristics and Challenges of IE

The characteristics of IE are qualities that differentiate an Intelligent Environment from traditional environments, and are discussed below.

1. Automation: the goal of any IE is to automate the usage of the devices within the environment. It is the defining characteristic of an Intelligent Environment.

2. Awareness: ability of the system to locate and recognize objects and people, their locations, and their needs.

3. Intelligence: allows the system to analyze the context, adapts to the people that live in it, learns from their behavior, and eventually recognizes as well as shows emotion.

4. Unobtrusive: provide unobtrusive and seamless human-machine interfaces. Better integration of technology into our environment, so that people can freely and inter-
actively use it. The occupants can talk to the environment using speech and natural language.

5. **Adaptable**: learn about the environment and the people within it in order to optimize their own behavior.

All this brings a collection of challenges in Intelligent Environments to be considered: home design and sensor layout, communication and pervasive computing, natural interfaces, management of available data, capture and interpretation of tasks, decision making for automation, robotic control, large-scale integration and inhabitant privacy.

### 2.2 ROS - Robot Operating System

Nowadays, processing power and storage are no longer restricting factors. So, in Intelligent Environments, the currently problem is the lack of a common middleware to interconnect heterogeneous distributed systems.

The successful transfer and application of a robotics middleware Player [7] in the domain of pervasive computing has been already shown [8]. The successor middleware, ROS (Robot Operating System) [9], is downward compatible w.r.t. existing drivers and includes many modern concepts of distributed architectures. And also has already been reported to have been used in the context of Intelligent Environments with more several benefits compared to other middleware [10, 11].

ROS was originally developed in 2007 under the name Switchyard by the Stanford Artificial Intelligence Laboratory in support of the Stanford AI Robot (STAIR) project. It is an open–source, meta–operating system for robots [12]. It provides the services you would expect from an operating system, including hardware abstraction, low–level device control, implementation of commonly–used functionality, message–passing between processes, and package management. It is based on a graph architecture where processing takes place in nodes that may receive, post and multiplex sensor, control, state, planning, actuator and other messages. Besides, the possibility of interconnecting several ROS servers, allowing to partition computation, easily supports developers of Intelligent Environments to build a more complex system in a Lego brick manner.

More details about the ROS concepts are explained below.

#### 2.2.1 ROS concepts

ROS has three levels of concepts [13]: the Filesystem level, the Computation Graph level, and the Community level.
Chapter 2 Background

Filesystem Level

The filesystem level concepts are ROS resources that you encounter on disk.

For organizing the ROS software, packages are the main unit. A package may contain ROS runtime processes (nodes), a ROS–dependent library, datasets, configuration files (.launch), or anything else. Inside the package it must be a Manifest (manifest.xml), to provide metadata about the package, including its license information and dependencies, as well as language–specific information such as compiler flags. The packages, in its turn, are organized in Stacks (collections of packages) that provide aggregate functionality. Stacks are also how ROS software is released and have associated version numbers. As in packages, the stack also has a manifest: Stack Manifests (stack.xml), which provide data about a stack, including its license information and its dependencies on other stacks. To define the data structures for messages sent in ROS there are Message (msg) types. And to define the request and response data structures for services in ROS there are Service (srv) types.

Computation Graph Level

The Computation Graph is the peer–to–peer network of ROS processes that are processing data together. The basic Computation Graph concepts of ROS are:

1. **Nodes**: Nodes are processes that perform computation. ROS is designed to be modular at a fine–grained scale; a robot control system will usually comprise many nodes. For example, one node controls the head, one node controls the wheel motors, one node performs localization, etc.

2. **Master**: The ROS Master provides name registration and lookup to the rest of the Computation Graph. Without the Master, nodes would not be able to find each other, exchange messages, or invoke services. Nodes communicate with the Master to report their registration information.

3. **Parameter Server**: The Parameter Server allows data to be stored by key in a central location. It is currently part of the Master.

4. **Messages**: A message is a simply a data structure, comprising typed fields. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types. Messages can include arbitrarily nested structures and arrays. Nodes communicate with each other by passing messages.

5. **Topics**: The topic is a name that is used to identify the content of the message. Messages are routed via a transport system with publish / subscribe semantics. A node sends out a message by publishing it to a given topic. A node that is interested in a certain kind of data will subscribe to the appropriate topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics. In general, publishers and subscribers are not
aware of each others existence. The idea is to decouple the production of information from its consumption. Logically, one can think of a topic as a strongly typed message bus. Each bus has a name, and anyone can connect to the bus to send or receive messages as long as they are the right type.

6. Services: The publish / subscribe model is a very flexible communication paradigm, but its many-to-many, one-way transport is not appropriate for request / reply interactions, which are often required in a distributed system. Request / reply is done via services, which are defined by a pair of message structures: one for the request and one for the reply. A providing node offers a service under a name and a client uses the service by sending the request message and awaiting the reply. ROS client libraries generally present this interaction to the programmer as if it were a remote procedure call.

7. Bags: Bags are a format for saving and playing back ROS message data. Bags are an important mechanism for storing data, such as sensor data, that can be difficult to collect but is necessary for developing and testing algorithms.

This architecture allows for decoupled operation, where the names are the primary means by which larger and more complex systems can be built. Names have a very important role in ROS: nodes, topics, services, and parameters all have names. Every ROS client library supports command-line remapping of names, which means a compiled program can be reconfigured at runtime to operate in a different Computation Graph topology.

Community Level

The main purpose of using ROS is to support code reuse in robotics research and development, encouraging to share and collaboration. For this reason, the ROS Community Level concepts are ROS resources that enable separate communities to exchange software and knowledge. These resources include: distributions, repositories, the ROS Wiki, Bug Ticket System, Mailings Lists, ROS Answers and the Willow Garage Blog [13].

2.3 Gazebo Simulator

In cooperation with the Player and Stage projects, Gazebo Simulator was initially created by Nathan Koenig and Andrew Howard at the University of Southern California. It is a multi-robot simulator for environments, incorporated into Willow Garage’s open-source Robotic Operating System (ROS). It is capable of simulating a population of robots, sensors and objects in a three-dimensional world. It generates both realistic sensor feedback and physically plausible interactions between objects.

To achieve this, Gazebo Simulator utilizes an open-source graphics renderer (Ogre [14])
and an open-source physics engine (ODE [15]) to model how the robot interacts with its surroundings, and to visualize both how the world appears and to generate renderings of what different sensors can see. Gazebo also uses the free open 3rd party library “Assimp” (Asset Import Library) [16] to import various well-known 3D model formats, so that the realistic reconstruction of real-world environments is possible.

By using standard ROS messages, higher-level client applications will not be aware if they are interacting with the real system or the simulated version. Developers will be able to develop applications that process the sensor data and command the simulated robot system to perform various manipulation tasks within the simulated world, and then easily have this software transitioned to run on the robot (almost without changing code).

2.4 URDF - Unified Robot Description Format

The Unified Robot Description Format (URDF) is an XML specification to describe a robot [17]. The main limitation is that only tree structures can be represented, ruling out all parallel robots. Also, the specification assumes the robot consists of rigid links connected by joints. The specification covers:

1. **Kinematic and dynamic description of the robot**: This part defines the basic physical properties of an object
2. **Visual representation of the robot**: This part defines how an object looks like in the simulation world.
3. **Collision model of the robot**: This part describes the isolated event in which two or more objects exert relatively strong forces on each other for a relatively short time.

The description of a robot consists of a set of link elements [18], and a set of joint elements [19] connecting the links together. But as mentioned above, only tree structures can be represented, and so the root element in a robot description file must be a robot, with all other elements must be encapsulated within.

The elements used to describe the robot are detailed below.

1. **Links**: stand for the basic building blocks of a model. Each body has an assigned mass, friction, bounce factor, and rendering features such as color, texture, etc. A link of an object can be represented in Gazebo Simulator by a rigid body. In physics, a rigid body is an idealization of a solid component of finite size in which the deformation is neglected, which means, the distance between any two points of the given rigid body remains constant, if external forces exerted on it. In classical mechanics, a rigid body is usually considered as a continuous mass distribution or in other words, a collection of point masses. Gazebo Simulator supports three basic shapes of a link: box, cylinder and sphere. Since the three basic geometry shapes are
not powerful enough to describe complex objects, the URDF language supports the model constructors directly import STL mesh files to represent the visual model as realistic as possible. Other formats as 3DS, OBJ, DAE are imported using the Open Asset Import Library [16], but testing this import feature showed that currently not all models with this formats are imported correctly.

2. Joints: help to connect bodies together to create dynamic relationships. There are a lot of joints, such as hinge joints for rotation along one or two axis, ball and socket joints, slider joints for translation along a single axis, and universal joints for rotation about two perpendicular joints. These joints can act as motors in addition the function of connecting two bodies. As long as there is a force to a joint, motion is produced because of the friction between the connected bodies. Nevertheless, there is one point should be paid attention on, is that both the model and simulation may be unstable if wrong parameters are taken when connecting many joints in a single model. The joint element describes the kinematics and dynamics of the joint. Two types of joints are commonly found in robots, and specifically on Intelligent Environments: revolute joints (e.g. for doors and windows), and prismatic joints (e.g. for drawers).

a) Revolute Joint: revolute joint provides single–axis rotation function used in many places such as door hinges and other uni–axial rotation devices. This type of joint is most frequently used in robotics.

b) Prismatic Joint: prismatic joint is a sliding joint that slides along one axis.

3. Extensions: send a ROS topic messages including the published information that can make the robot components coordinate with other collision objects. This element could also provide the means by which client programs can access and control models.

a) Transmission: Transmissions link actuators to joints. Multiple actuators may be linked to multiple joints through a complex transmission.

b) Sensor: A "sensor" on a robot is modeled by a link element. For simulation, a sensor is modeled with additional sensor-specific properties in gazebo elements.

c) Gazebo: The gazebo element is used for simulation. Sensors, controllers, materials for visualization... are modeled here with additional specific properties.

2.5 Sweet Home 3D - basis for a 3D CAD tool

The software developed in this thesis is an application based on Sweet Home 3D v3.0. According to its author, Sweet Home 3D is a free interior design application that helps you place your furniture on a house 2D plan, with a 3D preview. It is aimed at people who want to design their interior quickly, whether they are moving or they just want to redesign their
existing home. Numerous visual guides help you draw the plan of your home and layout furniture. You may draw the walls of your rooms upon the image of an existing plan, and then, drag and drop furniture onto the plan from a catalog organized by categories. Also Sweet Home 3D is extensible and lets users add 3D models of e.g. furniture, appliances, electronic devices, vehicles, plants, trees… any model in supported formats: OBJ, DAE, 3DS and LWS. Each change in the 2D plan is simultaneously updated in the 3D view, to show you a realistic rendering of your layout.

A useful documentation to learn about this software is the Sweet Home 3D API (Application Programming Interface). These documentation together with the source code helps to understand the structure and behaviour of the application. This is obligatory for programmers that want to modify code of an existing application.

Sweet Home 3D is based on a MVC (Model View Controller) architecture, so understanding how is organized its Model layer is essential. The Figure 2.1 presents almost all the classes and interfaces available in the version 1.5 of com.eteks.sweethome3d.model package that matches this Model layer.

A relevant section of the Sweet Home 3D code is the part that affects the 3D modeling and view. It uses Java3D [20], the extension to Java for displaying three dimensional graphics. It is important for us because the definition of the three dimensional coordinates orientation of the model is different from the coordinates orientation of the 3D view and also different from the coordinates definition on the Simulator Gazebo (see Figure 2.2). Therefore it will have to be considered when exporting the environment to URDF (to put the objects in the correct position and with the correct orientation).
Figure 2.1: UML diagram of com.eteks.sweethome3d.model package
(a) Model: Three dimensional Cartesian coordinate system with the \( z \)-axis pointing up and left-handed oriented

(b) 3Dview: Three dimensional Cartesian coordinate system with the \( y \)-axis pointing up and right-handed oriented

(c) Gazebo: Three dimensional Cartesian coordinate system with the \( z \)-axis pointing up and right-handed oriented

Figure 2.2: Three dimensional coordinates orientation
Chapter 3

Modeling and Simulation

In this chapter, not only the terms of modeling and simulation in general, but also in the field of robotics and Intelligent Environments are commented. That which is achieved with this is to justify why this software application has been developed, to prove that it is helpful and useful in this context.

First, we start with a short description of the general terms of modeling and simulation. And after, simulation in robotics is commented.

What is modeling?

Modeling is the process of producing a model; a model, as used in science, is a representation of the construction and working of some system of interest. The main goal of a model is to enable the analyst to predict the effect of changes to the system. A model is similar to but almost always simpler that the system it represents. A good model is a judicious tradeoff between realism and simplicity, because it should be as “real” as possible, but it should not be so complex that it is impossible to understand and experiment with it.

What is simulation?

In its broadest sense, simulation is the execution of a model over time to evaluate the performance of a system, existing or proposed, under different configurations of interest. While modeling targets the conceptualization, simulation challenges mainly focus on implementation.

Simulation is very important. Not all experiments can be done in the real world. Either they are too expensive or costly, too dangerous, are not easily repeatable or simply the system or product does not exist physically. But in the virtual world the model can be reconfigured and experimented with. Furthermore, often simulation (and models, too) deliberately emphasize one part of reality at the expense of other parts. Sometimes this is necessary due to computer power limitations. Sometimes it’s done to focus the analyst on an important aspect of the simulation.
There are two types of simulations, regarding the time: discrete event simulation in which the central assumption is that the system changes instantaneously in response to certain discrete events, and continuous simulation in which the simulator attempt to quantify the changes in a system continuously over time in response to controls.

Using simulation has several advantages, but I would like to emphasize these:

- Simulation is usually cheaper and safer than conducting the real experiment with the real thing.
- Let focus on a specific problem and develop&test new ideas.
- Simulations can often be conducted faster or slower than real time. This allows using them for efficient if–then–else analyses of different alternatives, speeds up the development process, reduces the time needed for iterative development and refinement, and save money.

3.1 Simulation in robotics

The importance of simulation technology increased due to, in part, the development of computers. And nowadays it is a powerful visualization, planning, and strategic tool in different areas of research and development, and robotics is no exception [21].

Robotics is a very interdisciplinary research field, as robots are introduced in various areas and used for many different operations [22]. Some of these disciplines are: operative systems, communications, image processing, concurrent and distributed programming, software engineering, task planning, logic reasoning, machine learning, multi–agent system, emergency...

Some advantages of using simulation, in general, has been commented in the previous section, but, particularly, using robotic simulation the most prominent issues are listed below:

1. It’s Easier. No need to understand all disciplines in robotics.

2. Detection of collisions.


4. The ability to visualize robot motion can quickly answer questions regarding path planning, workspace constraints and coordination issues with other systems.

5. Simulation helps us to understand what type of modifications and improvements should be made.

6. Allow to design and optimize robotic systems in advance, ahead of production.
7. Allow to test robot programs quickly, intuitively and in a safe environment.


In the context of Intelligent Environments, a middleware and simulator from the robotics domain is so feasible. The principal keys factor supporting these idea are:

- The challenges with respect to the heterogeneous devices and interfaces in robotics seem very similar to those found in the context of Intelligent Environments.
- Conceptually an Intelligent Environment is very similar to a static, non-movable robot, a "ImmoBot".

The use of simulation technology in Intelligent Environments is of particular importance to developers and researchers. Many of the required hardware technologies are not available at a reasonable price, and many of the application scenarios are being designed with the future in mind and well in advance of the hardware actually being available. Furthermore, the visualization tools in simulation are also very important in this context, because they can serve as a means to visualize received sensor data. For example, it can be evaluated how an environment "looks like" for a device that only has certain sensors, and allows the evaluation of cognition-based systems in simulated and real Intelligent Environments.

We have seen that simulation is very important in the context of robotics and Intelligent Environments. Therefore, the models must be made, and they are as important as the simulation. Usually this modeling task is cumbersome and very time-consuming. This goes worst if e.g. we need to simulate the behaviour of a robot in multiple environments because we would need a lot of time to model this different environments. Here is where develop a modeling software makes sense, and that is the reason why a prototyping toolkit for environments has been developed in this thesis.
Chapter 4

Designed application - Sweet Home 3D Extended

The designed application bases on Sweet Home 3D, and so it has the same features as his predecessor plus new ones. This chapter enumerates the new features of the developed software, the limitations and the requirements.

4.1 New Features

New features added to this application that Sweet Home 3D v3.0 does not have are explained below:

1. Added Plan > Add floor menu item to add a new floor at the top of the building with a copy of the walls and rooms of the actual plan view.

2. Added Plan > Delete floor menu item to delete the top floor of the building.

3. Added Tabbed Panes for the floor plans and for the furniture lists.

4. Added text fields and Locate button in Compass Panel to try to locate the geoposition by the postal address.

5. In Compass Panel when the Time Zone is changed the geolocation (latitude, longitude) as well.

6. Added 3D view > Export to URDF format... menu item to export the building and furniture to a set of URDF, OBJ, STL, LAUNCH, YAML, MATERIAL files organized in folders. On Appendix B – Sample this structure of files is shown and commented. The theoretically goal is what you see in 3D view (plus the light sources) is what you get when simulates in Gazebo.

7. Added fields in Room Panel to add additional information and the connections between rooms.
8. Added Plan > Export room connections to XML format menu item to export the information about connections between rooms to an XML file.

9. Added mass, center of mass (in bounding box limits) in furniture pieces.

10. Added Gazebo doors and windows Catalog.

11. Added Gazebo door or window furniture with a configurable revolute joint.

12. Added more fields in Modify furniture panel of gazebo door or window pieces to allow modify the revolute joint parameters.

13. Added checkbox in Import furniture panel to allow to import Gazebo door or window.

14. Export SVG file of selected floor.

4.2 Limitations and problems

Some features and contents of the original Sweet Home 3D v3.0 have been eliminated and others unchecked.

1. Unchecked features:
   a) Possibility to create and add Plug-ins.
   b) Print function.
   c) Video function.

2. Known problems/limitations:
   a) The value of mass is not shown correctly in the furniture table.
   b) When a door or window is attached to a wall, the angle is automatically added, but when modifies this piece it rounds the angle value and might not fit correctly with the wall direction, so is necessary to be moved again.
   c) Not all the elements of the URDF format are defined and editable on GUI, therefore to add or to modify them only is posible by hand after the export:
      i. The contact coefficients in collision element are not exported.
      ii. For joint elements: only revolute joint with the required limit element are declared and editable on GUI.
      iii. For the movable furniture: a default values of the inertia matrix in the inertia element are exported.
      iv. Default descriptions of transmission elements, gazebo controller elements and controller parameters (YAML files) are exported.
When the building is exported and visualized on Gazebo it shows some visual issues or imperfections. These are not actually bugs of the developed software but deficiencies or unfinished features of Simulator Gazebo. Probably related to the Open Asset Import Library, which Gazebo uses to import 3D model formats. This encountered visualization issues on Gazebo are:

1. Some 3D model formats are incorrectly spawned in Gazebo:
   a) 3ds: it shows fine the first group, but it spawn the next in the origin of the world
   b) obj: only spawns the first group or part of the model (with correct color or texture)
   c) dae: textures or colors not spawned and strange yellow and black colors spawned instead.

Due to this visualization problems all the models are converted to the STL format when the building is exported to URDF.

2. If the default colors of an object are not changed to one color before exporting to URDF, in Simulator Gazebo appears without colors due to the STL files have no associated material. Its material texture or color is added with a gazebo extension in the urdf file and defined in the SweetHome3DExtended.material script.

4.3 Requirements

The requirements of this software are almost the same as for the Sweet Home 3D, but to be able to run some of the added features is necessary some third party software:

1. The Postal Address Localization in the compass panel sends a request to the web http://nominatim.openstreetmap.org/, so it needs internet connection and the good operation of "nominatim".

2. The exporting URDF part needs the Meshlab application installed and the meshlab-server command in the path of the operating system to be able to convert automatically the exported meshes from OBJ format to STL format.
Chapter 5

Writing the application

This chapter describes how the development process was, enumerates the modified classes from the Sweet Home 3D v3.0 source code and also the new classes added.

5.1 Development Process

To achieve the desired behaviour and features, several chronologically steps were followed:

1. The developed application bases on Sweet Home 3D v3.0, and hence before starting modify classes or adding new ones were necessary to understand the architecture and behavior of this software. The web site [23], javadoc and the UML diagram of model classes of sweethome3d (Figure 2.1) were very helpful.

2. After understand the relations between the classes and their behaviours, the new structure could be designed adding the new concept and class floor and changing the related classes.

3. When the modification was finished, the features “add floor” and “delete floor” were implemented.

4. Then was the turn for the localization by postal address feature (This feature would be able to be added at any time).

5. To allow users export the environment to URDF, a new package called urdf was created. Almost all classes contained in this urdf package were generated by JAXB [24] from urdf xml schemas and modified after by hand. The urdf xml schemas wrote by myself were not completed but enough to generate the needed classes. Then, the most important class in this package was created: the Urdf class. It manages the export of the environment to the URDF format. Note that a lot of classes in this package are actually not used yet, but they would be useful for a future work.

6. Finally, the additional rooms information and the possibility to export it to an xml file were added. In this case the package xml was created by the same way as the
5.2 Modified Classes

Even though the source code is commented, there is a list of the almost all modified files and classes below. Each one with a brief comment about the changes made. This list is helpful to have an idea how many changes has been needed. Note that not all changes are listed and commented and also note that line numbers are approximate.

1. In sweethome3d package:
   a) package.properties: in lines 40,47,48 replaced Sweet Home 3D for Sweet Home 3D Extended
   b) SweetHome3DBootstrap: in line 49 added jaxb–impl–2.2.2.jar and in applicationpackages in line 75 added the package com.sun.xml
   c) SweetHome3D: replaced splashscreen with splashscreenextended

2. In applet package: changes added to avoid compiling errors (the applet functionality is not checked yet)
   a) ViewerHelper: in lines 293–299 added createTabbedFloorFurnitureView and createTabbedPlansView methods

3. In j3d package:
   a) Ground3D: replaced old home class for new floor class
   b) HomePieceOfFurniture3D: replaced old home class for new floor class
   c) PhotoRenderer: replaced old home class for new floor class and adapted to take the photo correctly with all floors
   d) Room3D: replaced old home class for new floor class
   e) Wall3D: replaced old home class for new floor class. In lines 318 328 361 465 637 660 added floorelevation and in lines 255–267 added code to erase the intersections between walls

4. In model package:
   a) Home: a lot of changes made
   b) HomeApplication: a lot of changes made
   c) Compass: in line 63 added this.postaladdress. In lines 443–453 setpostaladdress (now, when the timezone is changed, latitude and longitude as well). Added class PostalAddress at the end.
d) HomePieceOfFurniture: added mass, center of mass and isgazebodoororwindow

e) HomeFurnitureGroup: in line 126 setMass(1)

f) PieceOfFurniture: added mass, center of mass and isgazebodoororwindow

g) CatalogPieceOfFurniture: added boolean gazeboDoorOrWindow and floats
mass, centerOfMass

h) CatalogLight: in line 128 mass=1 in the method call

i) CatalogDoorOrWindow: added mass and centerOfMass as optional

j) Room: added information about connection between rooms: typeRoomInformation,
contentRoomInformation,fromXtoRoom, fromRoomToX

k) UserPreferences.properties: at the moment only english language supported

l) Wall: in line 607 limit = 2 * thickness/2 (now, each side of the wall has the half
of the thickness)

5. In swing package:

a) CompassPanel: in lines 91–100 new labels, in lines 372–641 createcomponent
labels city,country, street, zip code, in lines 750–x layout of new component
added

b) FileContentManager: in lines 245–262 urdf, in lines 291,380 urdf.xacro added

c) FurnitureTable: replaced old home class for new floor class, and added mass-
column

d) FurnitureTransferHandler: replaced old home class for new floor class

e) FurnitureCatalogTransferHandler: added homegazebodoororwindow

f) HomeComponent3D: a lot of changes made

g) HomeFurniturePanel: added mass, center of mass(x,y,z), effort, velocity, axis,
upperlimit, lowerlimit and verifyformat method

h) HomePane: a lot of changes made...

i) package.properties: added the text for the new labels

j) PhotoPanel: in line 946 added for loop to spawn all the floors of the building

k) PlanComponent: in lines 211,504,720 replaced old home class for new floor class

l) PlanTransferHandler: replaced old home class for new floor class

m) SwingViewFactory: in lines 89,97 replaced old home class for new floor class.
In lines 277–289 added createTabbedFloorFurnitureView and createTabbed-
PlansView
n) VideoPanel: changed PlanComponent planComponent to List<PlanComponent> planComponentsList because there will be more than one planComponent (one for each floor)

o) RoomPanel: added room connections information

p) ImportedFurnitureWizardStepPanel: added gazeboDoorOrWindow checkbox to import an actionable door or window

q) NullableSpinner: added setMinimum and setMaximum methods

6. In viewController package:

a) CompassController: replaced old home class for new floor class, added localization by postal address

b) ContentManager: in line 31 added URDF and XML in contentType

c) FurnitureController: replaced old home class for new floor class

d) HomeController: changed variables undosupport, undomanager, plancontroller, furniturecontroller to Lists to manage different floors. Added tabbedPlansController, tabbedFurnitureController, addFloor, gazeboDoorOrWindow... a lot of changes made

e) HomeController3D: adapted to manage all floors and to show camera icon in all plans

f) HomeFurnitureController: replaced old home class for new floor class. Added mass, centerOfMass, gazebo joint with its limits (upper, lower, velocity, effort) and axis of rotation

g) HomeView: added addFloor, deleteFloor, mass, export room connections, export to urdf, sort by mass, exportTourDF...

h) ImportedFurnitureWizardController: replaced old home class for new floor class. Added mass, center of mass, gazebo joint, gazebo door or window

i) LabelController: replaced old home class for new floor class

j) package.properties: in line 76 exportTourDF error and message, in line 613 sort and display mass

k) PlanController: replaced old home class for new floor class

l) ViewFactory: in lines 45-51 replaced old home class for new floor class, in lines 54–63 createTabbedFloorFurnitureView and createTabbedPlansView

m) WallController: replaced old home class for new floor class

n) HomeFurnitureController: added mass, center of mass, gazebo joints...
Chapter 5 Writing the application

7. In io package:
   a) DefaultTexturesCatalog.properties
   b) ContributedFurnitureCatalog.properties
   c) DefaultFurnitureCatalog.properties
   d) DefaultFurnitureCatalog: in line 177 added gazeboDoorOrWindow key, in lines 239–251 added mass, centerOfMassX, centerOfMassY, centerOfMassZ keys. In line 516 gazeboDoorOrWindow, in lines 536–539, 547–570 several changes, and added new sash (0.01f, 0.01f, 1, 0. . . to draw the 2D arc line in plan for a gazebo door or window
   e) FileUserPreferences: in line 106 furnitureGazeboDoorOrWindow, in lines 113–116 furnitureMass, centerOfMass. Method readFurnitureCatalog(Preferences preferences) in lines 439, 448–473 several changes. In method writeFurnitureCatalog(Preferences preferences) in lines 460, 637–651, 681–685 several changes

8. In tools package:
   a) operatingsystem.properties: adapted to Sweet Home 3D Extended

9. In plugin package:
   a) not revised

10. In resources package:
    a) splashscreenextended.jpg instead of splashscreen.jpg

5.3 New Classes

1. In model package:
   a) Floor: is the old Home class but with some modifications
   b) Location: (inside Compass class)
   c) CatalogGazeboDoorOrWindow
   d) GazeboDoorOrWindow
Chapter 5 Writing the application

e) HomeGazeboDoorOrWindow

2. In swing package:
   a) ControllerFloorAction
   b) TabbedFloorFurniture
   c) TabbedPlans

3. io.urdf package: All the package and classes inside are new.

4. In viewcontroller package:
   a) TabbedFloorFurnitureController
   b) TabbedPlansController
   c) TabbedView
Chapter 6

Conclusions

Based on the concepts of IE, a prototyping toolkit for robotics and Intelligent Environments has been developed in this work. The application allows to visually prototype new environments and to model existing environments. As is known, this modeling is a cumbersome and very time-consuming task. But with this software it becomes much more easier and quicker.

Also, the promising middleware for IE, ROS/Gazebo, has been briefly described. And the necessity of simulation in IE has been analized. All of that justifies the developed software in this thesis.

Besides, some tutorials have been made to show how to use the new features of the application. And to show the capabilities of this software, a part of the floor 1 of the building 0509 Neubau Innenhof (Wienandsbau) in the main campus has been modeled.

Even though the results of this research are very encouraging, obviously there are still some limitations. Most of the problems are described in the chapter “Writing the application“ section 4.2 “Limitation and problems”, but I would like to emphasize wich the problems related with the 3D formats importation would be: the issues of 3D formats are probably due to the Asset Import Library, some 3D formats are not well imported and that is why they are incorrectly spawned in Gazebo. This entailed a deterioration of the virtual appearance, because I was not able to spawn different colors for a STL model, neither to configure correctly the textures.

Future work can be planned to improve the application, to get a more realistic and sophisticated simulation world.

Firstly, as discussed in the previous part, we have problems with colors and textures of models. To solve this, the importing Gazebo issues of OBJ should be fixed. Then, the STL conversion would not be needed and so the source code of the software (only the Urdf class) should need a few modifications. Furthermore, if OBJ import is fixed, it will not be necessary to export all the walls in different models, it will be possible to export all of them as a unique file. And also, the script file (SweetHome3DExtended.material) where the materials are defined will not be needed.
And secondly, to facilitate even more the modeling task, could be added more editable fields and features in the toolkit like: the geolocation in a .world gazebo file, the contact coefficients, the other type of joints for the gazebo furniture, the values of inertia matrix for movable furniture... All this features makes the work easier because is usually more comfortable modeling something filling text fields than typing xml files.
Appendix A

HowTo - Tutorials

As Sweet Home 3D Extended is based on Sweet Home 3D, its manipulation is the same, and the user guide of Sweet Home 3D can be used as a tutorial (considering the unchecked features and errors mentioned earlier). However, as there are new features added, how to use some of them is explained below.

A.1 Import “Gazebo door or window”

The named Gazebo door or window is a door or window that will have a revolute joint and an associated controller when the building would be exported to URDF.

1. Design the model in two separated parts: movable part and unmovable part (sashes). Save them as one of supported formats OBJ, DAE, 3DS or LWS. And for both parts follow the next steps:

2. Click Furniture > Import furniture... and choose the model.

3. Put the correct orientation, it is important because the joint will be added always on the left side with down open direction as the top view of a Gazebo door image A.1 shows.

Figure A.1: Orientation of door or window
4. Check Gazebo door or window for the movable part, and check only Door or window for the unmovable part.

Now you have the Gazebo door or window in the Furniture Catalog, just drag and drop to add it in the floor.

**A.2 Modify the revolute joint in gazebo doors or windows**

The methods to show the Modify furniture panel are the same as in Sweet Home 3D, but when the piece of furniture is a Gazebo door or window, it shows as well the Revolute Joint configuration part. The values of upper Limit and lower Limit are automatically adapted when the shape is mirrored, but the next figure [A.2] should help you to know how this limits are defined.

![Figure A.2: Revolute Joint](image)

(a) Normal shape  
(b) Mirrored shape

**A.3 Export the content to URDF and launch it in simulator Gazebo**

Before export the building to URDF note that all pieces in a floor, that they are not below a room floor, will be exported as unmovable piece (except gazebo doors and windows). It means that they will be as walls.

For the other pieces (which there are below a room floor) you can select how to export them, as unmovable piece like the previous case, or as movable piece. The movable piece will have his own urdf file, and his launch file, so you will be able to spawn it in the simulation separated from the building.
To select unmovable just append _unmovable on the name field of the piece. To select movable do not append _unmovable. Note that if the mass value is 0, a value of 100kg will be added for unmovable pieces and 20 Kg for gazebo doors and windows.

Click 3D view > Export to URDF format. . . and the application exports a set of files organized in folders. This files are ready to simply be copied in the sweetgazebo package, then just launch the desired launch file (the launch file for the building, or the launch file of each available movable piece of furniture).

Also you have to add this lines in the manifest.xml file in the package sweetgazebo to get Gazebo to work with the correct materials and controller manager.

```xml
<depend package="pr2_controller_manager"/>
<depend package="gazebo"/>
<export>
  <gazebo gazebo_media_path="${prefix}" />
</export>
```

### A.4 Modify the exported files to improve the gazebo visualization

Due to problems described in the chapter “Designed application” section 4.2 “Limitation and problems” about the supported 3D formats in Gazebo, its necessary to modify the exported files if you want to improve the realism of the visualization of 3D models spawned in Gazebo.

It is posible to improve the visualization of objects that have only one face (as room floors, or a poster. . .): Convert the model of the object to 3ds format with Meshlab or other software, then change the filename parameter on the mesh attribute of the Link object in urdf.xacro building file and delete the material and GenTexCoord of the related gazebo extension.

```xml
<mesh filename="${find sweetgazebo) change this .stl to your .3ds" scale="0.01 0.01 0.01"/>
<gazebo reference=""object link"">
  <turnGravityOff>true</turnGravityOff>
  delete this line <material>object link</material>
  delete this line <GenTexCoord>true</GenTexCoord>
</gazebo>
```

I could not work out the reason, but sometimes the COLLADA (.dae) models are correctly spawned, so it might worth a try do the same process as for one face objects.
Appendix B

Sample

To show some results of using this toolkit, a part of the floor 1 of the building 0509 Neubau Innenhof (Wienandsbau) in the main campus was modeled.

First, the blueprint of the floor was added as background. Then all the walls and rooms were drewed. To make the environment more realistic, the office models (window, window frame, vmi Poster...) were designed on another software, saved as DAE format, and imported to Sweet Home 3D Extended. Finally, the model was exported to URDF.

The used office models are also released in [https://vmi.lmt.ei.tum.de/ros](https://vmi.lmt.ei.tum.de/ros).

The following file structure is the result of exporting to URDF, where *name* is the chosen name for the URDF "building" file when exported (see Figure B.1 for the 0509 building):

▷ *name*
  ▷ action_components
    ▷ launch – contains the controllers launch file of the action components
      ▷ controllers.launch – it is called by default by the building launch file
    ▷ Meshes
      ▷ MeshesOBJ – gazebo doors or windows models in OBJ format
      ▷ MeshesSTL – gazebo doors or windows models in STL format
    ▷ parameters – the yaml files with the parameters of the controller for each gazebo door or window
  ▷ MeshesOBJ – the rest of the models in OBJ format
    ▷ walls – the models of walls (each side of the wall in a separated OBJ file)
  ▷ MeshesSTL – the rest of the models in STL format
    ▷ walls – the models of walls (each side of the wall in a separated STL file)
▷ *name*_movable_Furniture
  ▷ launch – each movable model has his own launch file here
  ▷ MeshesOBJ – the movable models in OBJ format
  ▷ MeshesSTL – the movable models in STL format
  ▷ objects – urdf files describing each movable model
▷ Media
  ▷ materials
Appendix B Sample

▷ scripts
  ▷ SweetHome3DEndExted.material – script defining the materials for each component of the floor model
▷ textures – the textures of any component of the model are in this folder
▷ *name*.launch – the building launch file. It starts the world, sends the model urdf to param server, pushes model description to factory, spawn the robot on gazebo and loads controllers (including controllers.launch)
▷ *name*.urdf.xacro – the urdf file (robot description of the building)

Figure B.1: File structure of the 0509 building when exported as URDF

When the model is visualized and simulated on Gazebo it looks very close as looks on Sweet Home 3D. To show the results some screen shots of the visualization on Sweet
Appendix B Sample

Home 3D Extended $B.2 B.4 B.6 B.9$ and his simulation/visualization on Gazebo were taken $B.3 B.5 B.7 B.8 B.10 B.11 B.12$.

Figure B.2: Sweet Home 3D Extended: Floors
Figure B.3: Gazebo: Floor

Figure B.4: Sweet Home 3D Extended: Stairs
Figure B.5: Gazebo: Stairs

Figure B.6: Sweet Home 3D Extended: Corridor
Figure B.7: Gazebo: Corridor

Figure B.8: Gazebo: Opened door
Figure B.9: Sweet Home 3D Extended: Office

Figure B.10: Gazebo: Office
Figure B.11: Gazebo: Opened office window

Figure B.12: Gazebo: Movable furniture
# List of Figures

2.1 UML diagram of com.eteks.sweethome3d.model package ................. 15  
2.2 Three dimensional coordinates orientation .................................. 16  

A.1 Orientation of door or window ................................................... 31  
A.2 Revolute Joint ............................................................................. 32  

B.1 File structure of the 0509 building when exported as URDF ............ 35  
B.2 Sweet Home 3D Extended: Floors .................................................. 36  
B.3 Gazebo: Floor ............................................................................. 37  
B.4 Sweet Home 3D Extended: Stairs ................................................... 37  
B.5 Gazebo: Stairs ............................................................................. 38  
B.6 Sweet Home 3D Extended: Corridor ............................................ 38  
B.7 Gazebo: Corridor ....................................................................... 39  
B.8 Gazebo: Opened door ................................................................. 39  
B.9 Sweet Home 3D Extended: Office ............................................... 40  
B.10 Gazebo: Office ......................................................................... 40  
B.11 Gazebo: Opened office window ................................................... 41  
B.12 Gazebo: Movable furniture ....................................................... 41
Bibliography


