POLITECNICO DI MILANO
Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e Bioingegneria

Simulation of ambient sensors in a robotic home for elderly people

Relatore: Prof. Francesco Amigoni
Correlatore: Prof. Alessandro Saffiotti

Tesi di Laurea di:
Giacomo Mantenuto, matricola 765865

Anno Accademico 2012-2013

I would like to thank the Professor Francesco Amigoni, to have guided me through this thesis with his priceless advices. I would like to thank my correlator Professor Alessandro Saffiotti, for contributing with his expertise and making the process of writing the thesis joyful.
I thank my family for always being close and for always encouraging me to do my best and never give up. They are the ones that taught me about commitment and the satisfaction that comes from it. I also thank my girlfriend for her patience and her invaluable help and support since the beginning of this experience.

Abstract

In our society, where the population of elderly people keeps increasing rapidly and where medical progress leads to a considerably increased life expectancy, age-related diseases require constant medical assistance and solutions, both effective and cost aware need to be sought. In situations where patients cannot be anymore treated only in institutional setting, individual and personalized care in the patients' home environments plays an important role. Ambient assisted living solutions aim at applying ambient intelligence technology to support independent living of people with disabilities and special demands and to relieve the workload from family caregivers and health providers. Since ambient intelligence is designed for real-world and physical environments, effective use of sensors is vital. Without physical components that allow an intelligent agent to sense and act upon the environment, we would end up with theoretical algorithms with no practical use. With the availability of inexpensive low-power sensors these current "smart homes" are normally equipped with a large amount of networked devices, which collaboratively process the acquired data about the state of the environment as well as the activities and behaviours of its residents, detecting changes in their daily routines and providing proactive services. The implementation of new technologies is normally a complex, costly and a time consuming matter. Deploying sensors in a never before encountered world may cause unforeseen, and possibly negative side effects. For example, lighting conditions or reflective surfaces may all affect the way visual sensors operate. A strategy of online testing can be extremely slow and tedious. Time can be spent much more productively by testing and modifying sensors offline in preparation for the real experiments. The offline prototyping demands for a simulation environment and prototypes that allow us to selectively test acceptable solutions at an early stage, to address the particular needs of the different users. This thesis explains how to implement four ambient sensors such as a passive infrared motion detector, an optical switch, a light sensor and a pressure sensor, using the Robot Operating System (ROS) and its associated 3D simulation tool, Gazebo, in the three-dimensional model of a real testbed apartment (Ängen in Örebro, Sweden), before executing them in real life.

Sommario

Nella società odierna, dove la popolazione di anziani è in rapido aumento e il progresso medico porta ad un considerevole aumento della speranza di vita, le malattie legate all'età richiedono costante assistenza medica e la necessità di trovare soluzioni efficaci ed economiche. In situazioni dove i pazienti non possono più essere trattati in strutture sanitarie istituzionali in modo autonomo, l'assistenza individuale e personalizzata gioca un ruolo fondamentale. Soluzioni di ambient assisted living mirano ad applicare le tecnologie di intelligenza artificiale per sostenere la vita indipendente degli anziani o delle persone disabili, con lo scopo di alleviare il carico di lavoro da parte di familiari e operatori sanitari. L'uso efficace dei sensori è di conseguenza di vitale importanza. Senza componenti fisici che permettono ad un agente intelligente di percepire l'ambiente e di agire su di esso, ci ritroveremmo con algoritmi teorici di nessun uso pratico. Con la disponibilità di sensori economici e a basso consumo di potenza inoltre, queste "case intelligenti" sono normalmente dotate di una grande quantità di dispositivi interconnessi tra loro, che collaborativamente elaborano i dati acquisiti sullo stato dell'ambiente, sulle attività e i comportamenti dei suoi abitanti, rivelando cambiamenti nella loro routine quotidiana e fornendo servizi proattivi. L'implementazione di nuove tecnologie è normalmente un processo complesso, costoso e richiede molto tempo. L'applicazione di sensori in un mondo mai incontrato prima può causare imprevisti ed effetti collaterali spesso negativi. Le condizioni di illuminazione o superfici riflettenti possono per esempio influenzare il principio di funzionamento di un sensore ottico. Una strategia di test on-line può essere estremamente lenta e noiosa. Il tempo può essere speso in modo pi produttivo testando e modificando sensori e attuatori off-line, in preparazione agli esperimenti reali. Sono tuttavia richiesti un ambiente di simulazione e prototipi che permettano di testare selettivamente soluzioni accettabili in una prima fase iniziale, per affrontare le esigenze specifiche dei diversi utenti. Questa tesi spiega come implementare quattro sensori ambientali, quali un rilevatore di presenza, un interruttore ottico, un sensore di luce e un sensore di pressione, utilizzando il middleware noto come ROS (Robot Operating System) e il suo simulatore 3D, Gazebo, all'interno del modello tridimensionale di un vero e proprio appartamento installato nel quartiere residenziale di Ängen, in Örebro (Svezia), prima di essere installati nell'ambiente fisico reale.

Acknowledgments

This thesis has been developed at the Applied Autonomous Sensor System Centre (AASS) in Örebro University, Sweden. I would like to thank the staff of the centre.

List of Figures

Contents

Chapter 1

Introduction

Traditionally, elderly care has been the responsibility of family members and was provided within the extended family home. In modern societies, elderly care is now being provided by the state or charitable institutions. The reasons for this change include decreasing family size, the greater life expectancy of elderly people, the geographical dispersion of families, and the tendency for women to be educated and work outside the home [27].
Much research is being carried out on building ambient assisted living systems and most efforts are made in developing pervasive devices and use Ambient Intelligence to integrate these devices together in order to build up a safe environment around the assisted people and help them maintaining an independent living. Due to the advances in the miniaturization of electronics, computing devices with various capabilities and interfaces like sensors and actuators can now be purchased at very affordable prices. This technology can be networked and used with the coordination of intelligent software to understand the events and the relevant context of a specific environment and to take decisions in real-time or a posteriori, for example a system by alerting the family or the health providers in case of an anomalous occurrence.

1.1 Context: Ambient Assisted Living

Ambient Assisted Living (AAL) is a relatively new approach which promises to address special needs for elderly people or people with disabilities, reducing the cost of health and social care. Assisted living emerged in the 1990s as an eldercare alternative for people whom independent living is not secure but who do not need the 24-hour medical care provided by a nursing home and are too young to live in a retirement home. AAL aims at extending the time older people can live in their home environment by assisting them in carrying out activities of daily living by the use of advanced ICT technology and providing remote services including care services, eventually increasing their autonomy. The level of independence an elderly person living in his own home is related to his autonomy in performing the basic actions involved in daily living: to transfer to and from the bed, in and out of a chair, to move around the flat. In a more precise way, an assisted living resident is defined as a resident who needs assistance with at least one of the Activities of Day Living (ADLs), such as the basic self-care tasks or skills that people usually learn in their childhood, like feeding, bathing, dressing, walking or transferring for example from the bed to the wheelchair. For this aim housing facilities have been built, providing supervision or assistance with activities of daily living, coordination of services by outside health care providers, and monitoring of residence activities to help to ensure their health, safety and well-being, letting them age in place. The goal of AAL solutions is to apply ambient intelligence technology to enable people with specific demands to live in their preferred environment longer. The use of sensors is part of a package which can provide support for people with illnesses or people at risk of falling. Devices such as RFID, motion detectors and other smart sensors, are spatially distributed and built up in wireless sensor networks, cooperatively monitoring the environmental conditions in order to assist the daily activities of the patients. Monitors and detectors are often linked to a telecare system, which could be triggered when, for example, a person falls or has a seizure, or when gas or smoke are detected, so that appropriate help can be provided. Sensors can also detect problems as intruders or even help to prevent them. For example, a spoken reminder to turn off the cooker can help to prevent a kitchen fire, or a bed sensor that turns on the light when it detects a person getting out of bed in the night can help to prevent a fall. To give a numerical example, the Whole System Demonstrator headline findings after a trial involving 3154 patients, included these outcomes [26]:

- 45% reduction in mortality rates,
- 20% reduction in emergency admissions,
- 15% reduction in Accident & Emergency visits,
- 14% reduction in elective admissions,
- 14% reduction in bed days,
- 8% reduction in tariff costs.

AAL is specifically different from telemedicine and telehealth, because it does not deliver any health-related services nor provide clinical health care, but still the results are promising. The key concept is preventing and mitigating harm by reacting to unforeseen events and raising a help response promptly.

1.2 Motivations

We are moving into a new era of information and communication technology called Ubiquitous Computing. Ubiquitous sensing technologies allow us to elicit and collect information about steady or changing states of phenomena and circumstances, through a great number of distributed, heterogeneous sensors, actuators and processing components that exchange data and communicate and thereby form an ecology of information processing systems. Successful integration of sensor technology into everyday life depends on end users acceptance of the technology and how it affects their life. Privacy is the most common issue of sensor technology for smart homes. Most of the users do not like to carry a wearable device with them because it marks them as diseased or dependent. Patients do not want to wear medical equipment in public or even at home and even in life critical situations they sometimes decide against the implantation of a medical device due to their fear of a loss of independence [14]. In order to achieve a broad user acceptance it is necessary to take even hedonic aspects into account when it comes to the integration into existing home environments of sensor devices, which should be primarily invisible and passive. A good compromise of ubiquity, reliability and non-intrusiveness is established by the ambient sensors. Ambient sensors are sensors capable of jointly sensing multiple physical phenomena in surroundings, such as humidity, temperature, acoustic pressure, force, and light, and these measured parameters can be associated with events such as opening doors, human presence, tipping ladders, etc [29]. They avoid problems like wearing a big reader in case of RFID-based method and invasion of privacy in case of video camera-based method. Due to the complexity of new technologies, sensors and robotic subsystems are a complex, costly and time consuming matter. The inability to treat these individuals in their own surrounding due to safety issues, often force them to move to a new home. A way to efficiently prototype and develop ambient sensors including their physical and digital parameters is then required, and it demands for a simulation environment that allows to selectively investigate the characteristics and properties of those systems and to test them in different environments before installing them in the real ones. Specialized software is needed, but even in one of the best suited open source tool chain such as ROS [11](Robot Operating System) and its associated 3D simulator Gazebo [15],ambient sensors have not been implemented.

1.3 Objectives

The purpose of this thesis is to develop a toolkit to extend the Ängen Cognitive Environment (ACE) stack with the addition of ambient sensors to make the environment intelligent. The ACE stack is a specialized ROS stack developed at AASS Research Center at Örebro University to simulate cognitive environments, providing easy-to-use tools for building and testing actuators and robots in Gazebo.
The application developed in this thesis consists of a set of plugins that allow users to visually create and spawn ambient sensors such as passive infrared motion detectors, optical switches, pressure and light sensors, in any environment with multiple floors, walls, rooms, appliances, furnitures, doors, windows, lights and robots, giving the possibility to set parameters accordingly with different datasheets. The data gathered and processed by the sensors are made available through an interface according to the ROS communication paradigm, which every other sensor or robot can access, to retrieve information and react consequently. The plugins make extensive use of the sensors already implemented in Gazebo, without altering the original structure, not even the file organization of the ACE stack, enhancing the operating principle, in some cases simplifying it, with a focus on keeping the highest degree of simulation reproducibility and computational efficiency.

Chapter 2

Background and previous work

2.1 Ambient Intelligence and smart homes

This thesis takes place within the context of smart environments or so-called smart homes, with the goal to monitor the activities of elderly people, living at home, in order to continuously assess their degree of independence and therefore their autonomy. Today people live in homes that can be considered "intelligent" by 1960s standards, and for a very reasonable cost. Thermostats and movement sensors that control lighting are commonplace. Now the bar has moved much higher: even the ability to link movement sensors to a security alarm for detecting intruders will not impress a society which regularly interacts with such facilities. The basic idea behind Ambient Intelligence (AmI) is that by integrating an environment through the technology a system can be built as an "electronic butler", which senses features of the users and their environment and then reason about the gathered data, selecting actions to be performed.

2.2 Sensors

Ambient Intelligence has been characterized by researchers in different ways. All the definitions highlight the features that are expected in AmI technologies: sensitive, responsive, adaptive, transparent, ubiquitous and intelligent [12]. Ambient intelligence algorithms rely on sensors data from the real world. Without physical components that allow an intelligent agent to sense and act upon the environment, we end up indeed with theoretical algorithms that have no practical use. Sensors are used by software algorithms to perceive the environment and use this information to reason and decide the action that has to be taken to change the environment. Perception in accomplished using a variety of sensors. Sensors have been designed for position measurement, for detection of chemical and humidity sensing, and to determine readings for light, radiation, temperature, sound, strain, pressure, position, velocity and direction. Sensors are typically quite small and can be integrated into almost any AmI application. Three types of sensor technologies have been demonstrated ability to address the challenges of sensing human activity in a smart home, including wearable devices where sensor are worn by the residents, direct environment components where sensors are distributed in the environment, and infrastructure mediated system where sensors are installed on an existing home infrastructure. This thesis focuses on the simulation of the direct environment components. This technology typically consists of a set of sensors and an associated sensor network (wired or wireless) to transfer data to a centralized monitoring system where sensor fusion and activity inference take place.

2.2.1 Sensor technology used by smart homes

Many smart homes today adopt the concept of ubiquitous sensing where a network of sensors integrated with a network of processing devices yield a rich multi-model stream of data. The sensory data are analysed to recognize and monitor basic and instrumental activities of daily living performed by the residents such as bathing, dressing or preparing a meal. This approach allows smart homes to capture patterns possibly reflecting physical and cognitive health conditions and then recognize when the pattern begins to deviate from individualized norms and when atypical behaviour occurs that may indicate problems or require intervention [20]. One particular type of sensor that is commonly used in smart homes is the binary sensor, which simply detects the state of an object or movement with a single digit '1' or '0'. Various types of binary sensors have been used in smart homes including motion detectors, pressure sensors and contact or optical switches. Motion detectors and pressure sensors are usually used to detect occupant presence and locations throughout the house. Contact switches are usually installed on the doors in a smart home such as the front door and doors of cabinets and appliances to provide information on the specific interaction that the occupant performs with objects and appliances. The advantage of binary sensors include low cost, easy installation, with privacy concerns. However, these sensors only give information at an abstract level and are limited when inferring activities. It is for example impossible to understand which item is removed from a cabinet by simply knowing the state of the cabinet. A variety of other sensors can be deployed in smart homes to help infer activities and trigger automatic services. These sensors usually provide more specific information than simple binary sensors but give fewer details than video cameras for instance. Tracking and identifying people in the environment is an issue in AmI systems. If the location of a person is known, the system can serve the individual better by anticipating needs based on their preferences and delivering services based on when they are commonly required. The technology which is often used to track individuals are motion sensors. Motion sensors have been used as a backbone of security systems for decades. However, while they can detect movement they cannot provide information to distinguish who (or what) produced the movement. As an alternative, a person can wear a sensor that helps to track the individual. An example of this technology is RFID tags that can be coupled with an RFID tag reader to monitor the movement of the tagged objects.

2.2.2 Sensor data

There are efforts in the wide field of applications today to fuse heterogeneous data from sensors in order to provide outputs, which are more informative than the original data from particular sensors. Reliable data are essential to be able to fuse sensor data in sensor fusion system. Nevertheless, it is in some cases difficult or even impossible to build a real environment complete with deployed sensors in order to ensure reliable data for the research, development and testing of sensor data fusion systems. Implementation of a sensors simulation environment, which would generate synthetic sensor data (numerical and non- numerical) is a solution to fulfill such efforts.
Researchers have found that different types of sensor information are effective for classifying different type of activities. When trying to recognize actions that involve repetitive body motions (e.g. walking, running, sitting, standing, climbing stairs), data collected from motion detectors installed in a strategic position have been used. In contrast, other activities are not as easily distinguishable by body position. In these cases, researchers observe the smart home residents interaction with objects of interest such as doors, windows, refrigerators, keys and medicine containers, through pressure sensors and optical switches.

2.3 ROS - Robot Operating system

Robot Operating System (ROS) is an open source software framework for robot software development, originally developed in 2007 under the name *switchyard* by the Stanford Artificial Intelligence Laboratory in support of the Stanford AI Robot (STAIR) project [22]. It provides operating system-like functionalities including hardware abstraction, lowlevel device control, messagepassing between processes and package management. Tools and libraries are also available for building and running the code across multiple computers. 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. This runtime graph is a peer-to-peer network of processes, coupled using the ROS communication infrastructure. ROS implements several different styles of communication, including synchronous RPC-style communication over services, asynchronous streaming of data over topics, and storage of data on a Parameter Server [11]. ROS currently only runs on Unix-based platforms. The library and the core system, along with useful tools and libraries, are regularly released as a Linux-like distribution that provides a set of compatible software for others to use and build upon. Even if it is not a realtime framework, it is possible to integrate ROS with realtime code. More details about the ROS goals and concepts are explained below.

2.4 Goals

The primary goal of ROS is *not* to be a framework with the most features, rather it is to support code reuse in robotics research and development. ROS is a distributed framework of processes that enables executables to be individually designed and loosely coupled at runtime. These processes can be grouped into Packages and Stacks, which can be easily shared and distributed. ROS also supports a federated system of code Repositories that enable collaboration to be distributed as well. This design, from the filesystem level to the community level, enables independent decisions about development and implementation, but all can be brought together with ROS infrastructure tools.
In support of this primary goal of sharing and collaboration, there are several other goals of the ROS framework [23]:

- Thin: ROS is designed to be as thin as possible so that code written for ROS can be used with other robot software frameworks. A corollary to this is that ROS is easy to integrate with other robot software frameworks: ROS has already been integrated with OpenRAVE, Orocos, and Player.

- Peer-to-peer: a system built using ROS consists of a number of processes, potentially on a number of different hosts, connected at runtime in a peer-to-peer topology.
- Tools-based: in order to manage the complexity the developers opted for a microkernel design, where a large number of small tools are used to build and run the various ROS components, rather than constructing a monolithic development and runtime environment. These tools perform various tasks, e.g. navigate the source code tree, get and set the configuration parameters, visualize the peer-to-peer connection topology, measure bandwidth utilisation, graphically plot message data, auto-generate documentation, and so on.
- Multi-lingual: ROS currently supports four very different languages such as C++, Python, Octave, and LISP, when other language ports in various states of completion.
- Easy testing: ROS has a builtin unit/integration test framework called rostest that makes it easy to bring up and tear down test fixtures.
- Scaling: ROS is appropriate for large runtime systems and for large development processes.
- Free and Open Source: the full source code of ROS is publicly available and it is distributed under the terms of the BSD license, which allows the development of both non-commercial and commercial projects.

2.5 Concepts

The fundamental concepts of the ROS implementation are nodes, messages, topics, and services. Nodes are processes that perform computation. ROS is designed to be modular at a fine-grained scale: a system is typically comprised of many nodes. Nodes communicate with each other by passing messages. A message is a strictly typed data structure. Standard primitive types (integer, floating point, boolean, ect.) are supported, as are arrays of primitive types and constants. Message can be composed of other messages, and arrays of other messages, nested arbitrarily deep. A node sends a message by publishing it to a given topic, which is simply a string. 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 simplest communications are along pipelines. However, graphs are usually far more complex, often containing cyrcles and one-to-many or many-to-many connections. Although the topic-based publish-subscribed model is a flexible communication paradigm, its "broadcast" routing scheme is not appropriate for synchronous transactions, which can simplify the design of some nodes. In ROS this is called a service, defined by a string name and a pair of strictly typed messages: one for the request and one for the response. This is analogous to web services, which are defined by URIs and have request and response documents of well-defined types. Note that, unlike topics, only one node can advertise a service of any particular name.

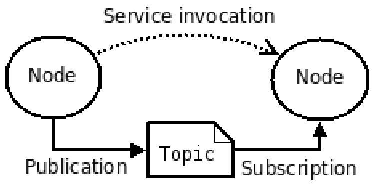


Figure 2.1: Basic concepts of the communication between nodes.

2.6 URDF - Unified Robot Description Format

Each visual entity in the environment needs to be described in a way that ROS will recognise it. For describing them it is necessary to make use of a XML language called URDF (Unified Robot Description Format). The main limitation at this point is that only tree structures can be represented, ruling out all parallel robots. Also, the specification assumes the sensor consists of rigid links connected by joints; therefore flexible elements are not supported. The specification covers:

- Kinematic and dynamic description of the robot
- Visual representation of the robot
- Collision model of the robot

Bodies represent the basic building blocks of a model. Their physical representation take form of geometric shapes chosen from boxes, spheres, cylinders, planes and lines. Each body has an assigned mass, friction, bounce factor, and rendering properties such as color, texture, transparency, etc. Joints provide the mechanism to connect bodies together to form kinematic and dynamic relationships. A variety of joints are available including hinge joints for rotation along one or two axis, slider joints for translation along a single axis, ball and socket joints, and universal joints for rotation along to perpendicular joints. Besides connecting two bodies together, these joints can act like motors. When a force is applied to the joint, the friction between the connected body and the other bodies cause motion.

2.7 Gazebo simulator

Simulation tools play a significant role in design and development of complex robotics systems. They allow to implement applications that does not depend physically on the actual device, saving cost and time. Modelling and simulation of rigid robots has been relatively matured over the past two decades, however in order to model the state of the art in robotics with new sensors and actuators and advanced control algorithms with realistic physics, there is a need to have improved simulation tools. Gazebo is a multi-robot simulator for outdoor and indoor environments, created by Nathan Koenig and Andrew Howard at the University of Southern California, and incorporated into the above mentioned ROS. Gazebo is capable of simulating a population of robots, sensors and objects, and does so in a three-dimensional world. It generates both realistic sensor feedback and physically-plausible interactions between objects (it includes an accurate simulation of rigid-body physics). Since Gazebo realistically simulates robots and environments, code designed to operate a physical robot is normally executed on an artificial version in order to avoid common problems associated with hardware such as short battery life, hardware failures, and unexpected and dangerous behaviours. It is also faster to spin up a simulation engine than continually run code on a robot, especially when the simulation engine can run faster than real-time. Gazebo Simulator utilizes an open-source graphics renderer (Ogre[6]) and an open-source physics engine (ODE [5]) 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 [4]) to import various well-known 3D model formats, so that the realistic reconstruction of realworld environments is possible. By using standard ROS messages, higherlevel 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 the code).

2.7.1 Architecture

A major feature of Gazebo is the ability to easily create new robots, actuators, sensors, and arbitrary objects. As a result, Gazebo maintains a simple API for addition of these objects, called models, and the necessary hooks for interaction with client programs. A layer below this API resides the third party libraries that handle both the physical simulation and visualization. This architecture is graphically depicted in Figure 2.2.

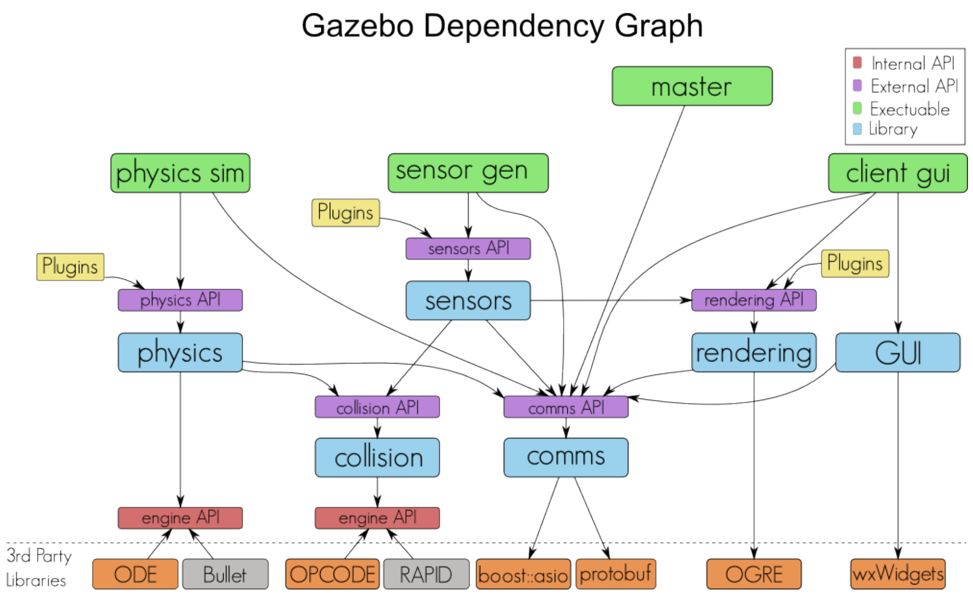


Figure 2.2: Gazebo Dependency Graph

The World represents the set of all models and environmental factors such as gravity and lighting. Each model is composed of at least one body and any number of joints and sensors. The third party libraries interface with Gazebo at the lowest level. This prevents models from becoming dependent on specific tools that may change in the future. Finally, client commands are received and data returned through a shared memory interface. A model can have many interfaces for functions involving, for example, control of joints and transmission of sensor data.

2.7.2 Physics Engine

The Open Dynamic Engine[8], created by Russel Smith is a widely used physics engine in the open source community. It is designed to simulate the dynamics and kinematics associated with articulated rigid bodies. This engine includes many features such as numerous joints, collision detection, mass and rotational functions, and many geometries including arbitrary triangle meshes. Gazebo utilizes these features by providing a layer of abstraction situated between ODE and Gazebo models. This layer allows easy creation of both normal and abstract objects such as layer rays and ground planes while maintaining all the functionality provided by ODE.

2.7.3 Visualization

A well designed simulator usually provides some form of user interface, and Gazebo requires one that is both sophisticated and fast. The heart of Gazebo lies in its ability to simulate dynamics, and this requires significant work on behalf of the user's computer. A slow and cumbersome user interface would only detract from the simulator's primary purpose. To account for this, OpenGL and GLUT (OpenGL Utility Toolkit)[9] were chosen as the default visualization tools. Besides ROS has another 3D visualization tool: Rviz. It can analyze collisions and represent them as a three-dimensional model, show the information about joints, mass and center of mass. It can create a new rendering window with an image and show data from a laser scan, with different options for rendering modes, accumulation, etc. Although also Rviz is a powerful tool that allows not only to reproduce sensor data but to control the sensors themselves, non of them excludes the other, but it is useful to use both of them in a complementary way. The advantage of Rviz is that the user can choose the data he wants to reproduce and personalize the interface. It is convenient for example when the user wants to lighten the simulation. For example, displaying the rays of a laser scan in Gazebo may slow down the simulation, so visualizing them in Rviz could solve the problem.

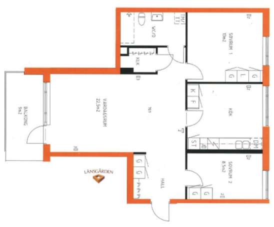
2.8 Sensor used in simulation

A complete environment is essentially a collection of models and sensors. The ground and buildings represent stationary models while robots and other objects are dynamic. Sensors remain separate from the dynamic simulation since they only collect data, or emit data if it is an active sensor. A sensor in Gazebo is an abstract device lacking a physical representation.

It only gains embodiment when incorporated into a model. This feature allows for the reuse of sensors in numerous models thereby reducing the code and confusion. There currently are three sensor implementations including an odometer, ray proximity, and a camera. Odometry is easily accessible through integration of the distance travelled. The ray proximity sensor returns the contact point of the closest object along the ray's path. This generic sensor has been incorporated into a Sick LMS200 model to simulate a scanning laser range finder, and also a sonar array for the Pioneer 2. Finally, the camera renders a scene using OpenGL from the perspective of the model the camera is attached to. Currently the camera sensor is used for both a Sony VID30 camera and the "God's eye" view of the world [18].

2.9 Ängen testbed facility model

The Ängen facility is a newly built senior living facility that provides an extensible, fully functional research facility for the development of assistive technology, with a particular emphasis on AmI solutions for elderly care, implemented in the centre for Applied Autonomous Sensor System (AASS) research group at Orebro University. The Ängen facility has been built and is administered by the Länsgården real estate company and began to host residents in Spring 2011. Ängen aims to build an open infrastructure for assessing deployability and validation of research prototypes, involving several partners representing complementary communities around elderly care. The Ängen apartment includes some robots and many simple devices - like people detectors and pressure sensors under the chairs, optical switches attached to doors and shutters as well as a special infrastructure that includes RFID-based sensory technology to provide reliable localization for robots and people through a special floor. These robots and devices may cooperate to generate some sort of collective intelligence.


(a)

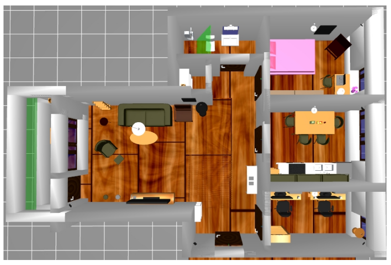
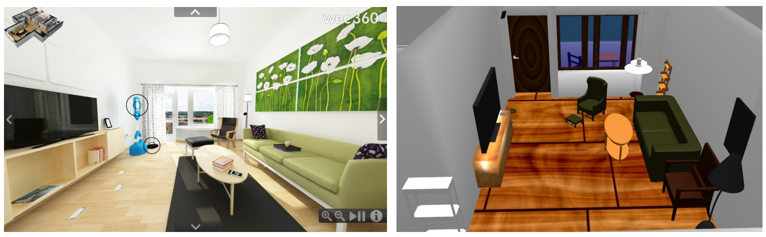
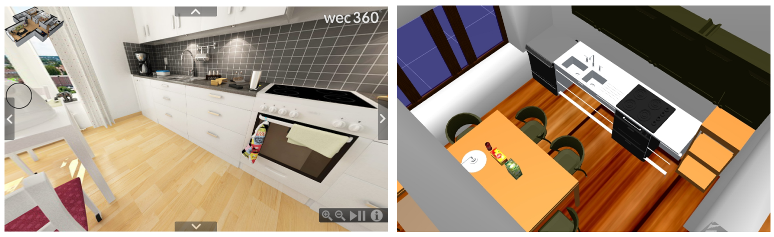

(b)

Figure 2.3: Floor plan of the real apartment (a) and top view of the simulated model (b).



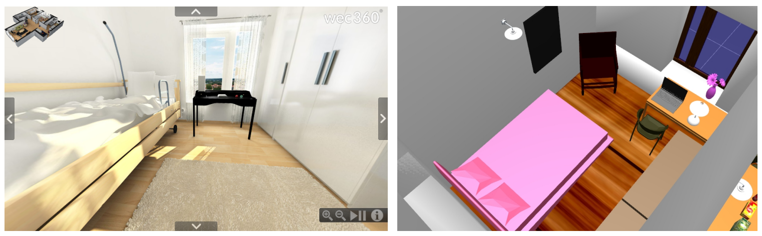
(a) (b)

Figure 2.4: Views of the real (a) and the modelled (b) livingroom.



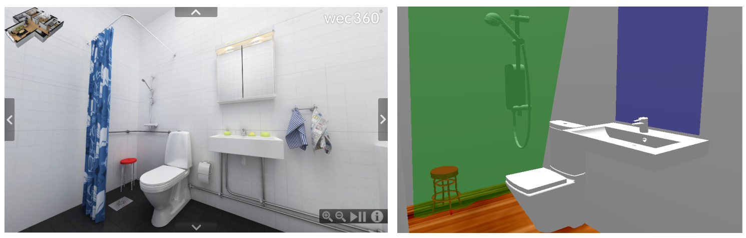
(a) (b)

Figure 2.5: Views of the real (a) and the modelled (b) kitchen.



(a) (b)

Figure 2.6: Views of the real (a) and the modelled (b) bedroom.



(a) (b)

Figure 2.7: Views of the real (a) and the modelled (b) bathroom.

Chapter 3

System and package overview

The purpose of this thesis is to implement in ROS (and in Gazebo) ambient sensors such as motion detectors, optical switches, light sensors and pressure sensors, not implemented yet in the framework. The development of those sensors takes the ones already present in ROS, such as cameras, lasers and contact bumpers, as a starting point, extending the functionalities, sometimes simplifying them, and manage to make them communicate and integrate with the environment. The approach is to use plugins. A plugin is a code snipplet that is compiled as a shared library and inserted into the simulation. The plugin has direct access to all the functionality of ROS through the standard C++ classes, documented in the public API. Using plugins is very useful. They let the developer directly control joints and links of the model as well as data generation and processing of the attached sensor, are self contained routines easily shared that can be inserted and removed at run-time without any overhead of the transport layer (e.g. no serialization and deserialization of messages). They are useful in many use cases, from data filtering to noise models addition. These advantages make plugins more flexible and so preferable to client nodes. Launching few sensors with the incorporated plugin is better than firstly spawning the decoupled models and then execute a node that have to access them. Every node that is executed requires in average 10% of the CPU within the simulation. Considering that the simulator requires around 60% of the CPU, it may be irrelevant for a test with only few sensors but it becomes inconceivable if we want to simulate a real environments with more than ten sensors, especially if we add the presence of robots.

3.1 Angen Sensor Plugins Package

Angen Sensor Plugins is a package with the aim to expand the Angen Cognitive Stack (ACE), a specialized ROS stack developed at AASS Research Center at Örebro University that provides easy-to-use tools for building and simulating cognitive environments in ROS Gazebo, with the addition of sensors to make the environment intelligent. The core of the package is a set of controllers that take the gazebo ones as starting point and are modified accordingly with the functional and operational requirements of the environment. The sensors implemented are four: passive infrared motion detector, optical switch, pressure sensor, photosensor (or photodetector). The package contains the URDF models as well as the 3D meshes and textures of the sensors, headers and source files, custom messages used by the controllers to publish sensor information and some configuration files.

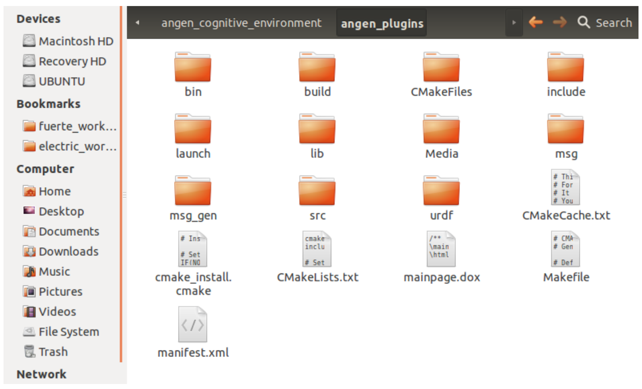


Figure 3.1: Package overview

The motion detector is implemented using a rangefinder laser in order to overtake the impossibility to simulate the infrared emitting light in Gazebo and simplifying by far its derived operative principle. The laser is tilted and the gathered range values are store in a matrix; the difference with the previous scans is compared with a threshold to detect movements. The optical switch makes use once again of a laser, with only one ray. A small tag is attached to doors and shutters, so when the door is closed the small tag breaks the ray and the event is triggered. The pressure sensor is a bumper that detects collisions and measure the z-axis component of the opposite ground reaction force exerted by the floors. The light sensor is a camera that computes a one-pixel image and calculate the RGB color space luminance. In the following chapters every sensor is explained in details.

Chapter 4

Passive infrared motion detector (PIR)

Figure 4.1: Passive infrared motion detector.

The role of the popular passive infrared motion detector has been reviewed during the last years, from a simple peripheral component of security systems to a basic building block in the integrated building automation systems[17]. Originally used to protect objects or rooms from unwanted intruders, they are now employed in daily activity recognition systems for eldery people in the smart environments discussed before. Therefore beyond the common requirements of security and access control, the occupancy picture of the building is extremely helpful to obtain activity profiles of patients, to provide a warning sign for unhealthy cases such as skipping meals and for the intervention during security incidents.

In Ambient Assisted Living, motion detector sensors are deployed differently than in the typical security applications and different results are expected by the users from the sensors. In security applications, it is required to inform the security system as soon as possible about any movement in the observed area, while in AAL applications the system and at the end the users are interested in the intensity of the movement in a given time frame (typically from several seconds to a minute) and everything has to be developed using a minimal number of sensors without affecting the patients behavior or daily routine. There are several advantages using PIR sensors. First, they do not require any signal or device on the object to be tracked. Second, they can work in dark environment as well, whereas vision-based system cannot, and third, they are cheap, easy to use and they not require huge computational power.
Before entering the details of how the sensor works and which is the operative principle, it is appropriate to describe in few words the context in which it is placed and what is its real purpose: motion detection. Motion detection is the process of detecting a change in position of an object relative to its surroundings or the change in the surroundings relative to an object, and can be achieved by both mechanical and electronic methods. The most basic forms of electronic motion detection are acoustical detection and optical detection. Optical motion detection devices, such as the passive infrared sensors implemented in this thesis, have a sensor that detects a disturbance in the infrared spectrum, such as a person or an animal. Once detected, an electronic signal can activate an alarm or a camera that can capture an image or video of the motioner.

4.1 How the real sensor works

Strictly speaking, individual PIR sensors do not detect motion; rather, they detect abrupt changes in temperature at a given point. The term passive in this instance refers to the fact that PIR devices contribute no energy of their own - do not generate or radiate any energy for detection purposes - but they work entirely by detecting the energy given off by other objects [24]. For this reason, many PIRs can be used together within the same area without interfering with each other.
All objects with a temperature above absolute zero emit heat energy in the form of optical radiation (light). Usually this light is invisible to the human eye because body temperature radiates at infrared wavelengths, but it can be detected by electronic devices designed for such a purpose. As an object, such as a human, passes in front of the background, such as a wall, the

temperature at that point will rise from room temperature to body temperature, and then back again. This quick change triggers the detection.
The sensor technology has evolved from simple thermistor sensors to highly sophisticated differential pyro-electric devices with microprocessor based signal processing [10]. The most common models have numerous Fresnel lenses or mirror segments, an effective range of about ten meters, and a field of view less than 180 degrees (models with wider fields of view, including 360 degrees, are typically designed to mount on a ceiling).

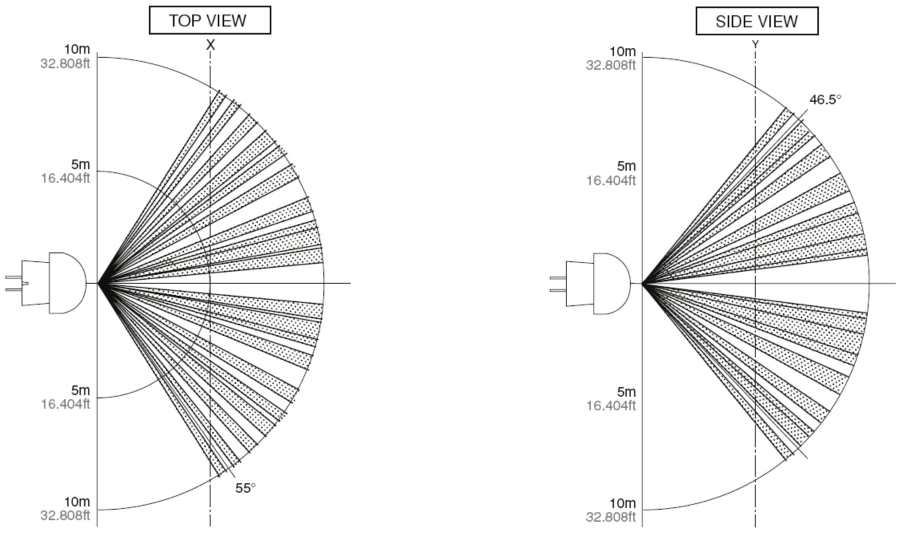


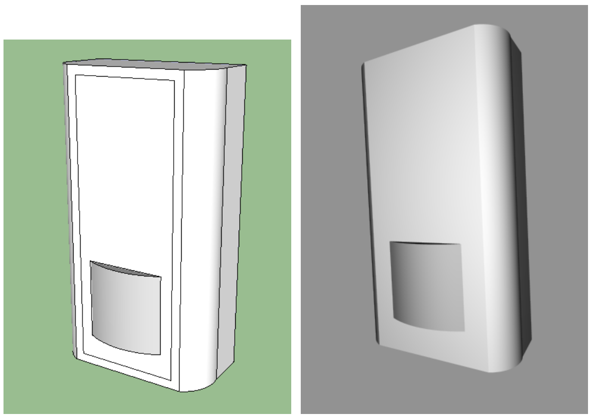
Figure 4.2: Example of horizontal and vertical view of detection zones of the PIR sensor

The occupancy monitoring system consist of a transmitter unit, a receiver unit and a commercially available passive infrared detector (PIRD). The microcontroller in the transmitter unit samples the output of the motion sensor via its analog to digital converter at a rate of 20 Hz. This digitized data is then subject to a motion detection algorithm embedded in the flash memory of the microcontroller, which determines whether activity has occurred or not. When motion is deemed to be of significance, the relevant data is time stamped by accessing the real-time clock and sent to the wireless transceiver module for transfer to the receiver unit. The selection of an analog infrared sensor provides the ability to set a minimum threshold to increase the detector sensivity, wider detection area and such a device consumes very low power [16].

4.2 How the simulated sensor works

Since it is not possible to simulate the emitting heat energy of the objects in Gazebo, the detection is triggered by the change of the ranges values gathered by a laser rangefinder. A laser rangefinder is a device which uses a laser beam to determine the distance to an object.

4.2.1 Design



(a) Sketch-up format *(b) Collada file*

Figure 4.3: Three-dimensional model of the motion detector

The model consists of two links and one joint. The first more simplistic version only included one link and no joints. In order to make it more real a link to simulate the housing is added to a spherical link that acts as the core of the sensor. Therefore a joint to connect them is required. The joint is a hinge joint that rotates along the axis and has a limited range specified by the upper and lower limits. The code below shows an example of the plugin:

```
<sensor name='pir_sensor' type='ray' always_on='1'
        update_rate='10' visualize='false'>
  <ray>
    <scan>
      <horizontal samples="120" resolution="1"
                  min_angle="-60" max_angle="60" />
    </scan>
    <range min="0.05" max="4" />
  </ray>
  <plugin name="angen_pir" filename="libangen_pir.so">
    <gaussianNoise>0</gaussianNoise>
    <alwaysOn>true</alwaysOn>
    <updateRate>10</updateRate>
    <topicName>room_state</topicName>
    <frameName>link1</frameName>
    <fov>90</fov>
    <fovRes>10</fovRes>
  </plugin>
</sensor>
```

The link element describes a rigid body with an inertia, visual features and collision properties. The visual element specifies the shape of the object (box, cylinder, etc.) for visualization purposes. The geometrical shape of the visual object is described by a trimesh, and an optional scale that scales the mesh's axis-aligned-bounding-box. The trimesh formats accepted by Gazebo are .dae (collada), .stl (stereo-lithography) and .mesh (Ogre mesh). In this thesis the models are created with the Google-Sketch-Up 3D modelling software [3], or downloaded from the online repository Google 3D Warehouse [2]. The collision properties are similar to the visual properties of a link, but can be different. For example, simpler collision models are often used to reduce computation time. The joint element describes the kinematics and dynamics of the joint, including the axis of rotation, and also specifies the safety limits of the joint, particularly useful in simulation. The plugin, in order to work correctly, needs a ray sensor as its parent. The sensor element describes basic properties of the visual sensor such as the name, the type and the frequency at which the sensor data is generated. Other attributes contribute to characterize the sensor, like the pose of the sensor optical frame, relative to the sensor parent reference frame (the sensor optical frame adopts the conventions of z-forward, x-right and y-down), the horizontal samples, the number of simulated rays to generate per complete laser sweep cycle and their resolution (this number is multiplied by samples to determine the number of range data points returned), the angles of the first and the last range measurement in radians and the smallest and the largest distances that can be measured. Although is possible to set attributes to get also a vertical scan, samples are collected only horizontally to gain major flexibility of the data structure and the operations on it, flexibility that may be lost working on a three-dimensional point cloud. In simulation large sensors will slow down overall performance. Depending on update rates required, it is then recommended to keep the ray resolution

and update rates as low as possible.
For that concerns the plugin itself the most important parameters are the filename and the update rate: the first is the name of the library generated from the compiled source file and the rate is the rate the plugin is throttled and should be the same of the sensor update rate in order to be synchronized with the scans. For simplicity and bigger precision the gaussian noise is removed (set to zero). Another important attribute, common to all the sensor plugins, is the name of the topic the sensor will send data on. The plugin is similar to the GazeboRosLaser controller, but requires two additional parameters: field of view and field of view resolution (fov and fovRes respectively).

4.2.2 Implementation

As mentioned before the plugin requires a ray sensor as its parent. This sensor casts rays into the world, test for intersections, and reports the range to the nearest object. The scans are taken in a counter-clockwise direction. Angles are measured counter clockwise with 0 pointing directly forward. The plugin stores the gathered ranges values in a matrix. Then the laser is tilted, making the spherical link rotate of an angle equal to the resolution of the horizontal field of view. The tilting is implemented as a call to the gazebo *SetModelConfiguration* service, which allows to rotate the joint along the rotation axis of an angle passed as parameter, without invoking dynamics. The number of scans is given by the ratio between the vertical field of view and its resolution. The joint positions are calculated and stored in another array. Every second the sensor executes as many scans as the update rate, and every scan at time t is compared with the prior scan of the same angle, at time t-1. A threshold is set to trigger the movement, so when the value of the difference between the two sequential scan is bigger than the threshold the boolean field of the message is set to True.

The message published by the laser sensor is showed below:

```
std_msgs/Header header
float32 angle_min
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan_time
float32 range_min
```

```
float32 range_max
float32[] ranges
float32[] intensities
```

while the PirMsg.msg is:

```
std_msgs/Header header
float32 angle
bool moved
```

where *header* contains the time stamp of the current scan and the frame name of the PIR sensor currently being listened to, *angle* contains the angle in radians within the field of view the sensor is scanning at the time stamp, and *moved* tells if any object within the field of view moved or not at that instance.

An example of the message output can be displayed on a new terminal typing the following command:

```
rostopic echo /pir_sensors/bedroom_pir/room_state
```

It will print:

```
header:
  seq: 0
  stamp:
    secs: 58
    nsecs: 235000000
  frame_id: /link1
angle: 0.698131680489
moved: False
```

4.3 Example

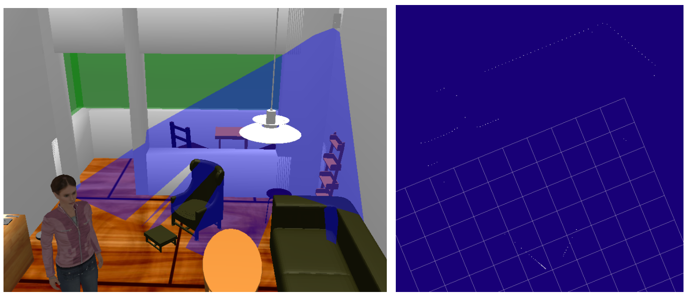
Many of the sensors implemented in this thesis have been initially tested within an empty world, but simulating a laser range finder with no walls or objects would not make sense. At the beginning the motion detectors were created inside the model of the empty apartment, in order to get feedbacks about the overall functioning. At the later stage every single piece of furniture was added to the scene to set positions, angles and other parameters to properly calibrate the sensors, accordingly to the furniture placement. To start the simulation with the empty environment a launch file is used:

```
<launch>
  <node name="gazebo" pkg="gazebo" type="gazebo"
        args=$(find angen_gazebo)/worlds/angen_empty.world
        respawn="false" output="screen"/>

  <param name="pir_description" command=$(find xacro)/xacro.py
        '$(find angen_plugins)/urdf/pir.sdf' />

  <node name="$(spawn_node)" pkg="gazebo" type="spawn_model"
        args=-gazebo -param /pir_description
        -x -1.6 -y -4.65 -z 2.9 -Y 2.55 -model test_pir
        respawn="false" output="screen" />
</launch>
```

As soon as the model is spawned into the world the plugin is loaded. Subsequently the matrix of the range values is initialized and like so the joint positions are calculated relatively to the field of view and its resolution. The field of view of the motion detector looks like beams reaching out into an area and stopping at the nearest solid object, like a wall, floor, human body or piece of furniture (Figure 4.4).

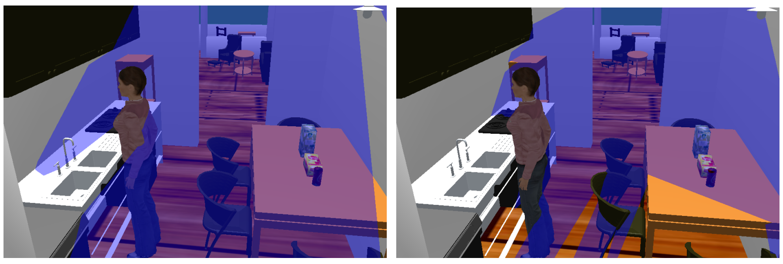


(a) A scan of the living room *(b) Rviz*

Figure 4.4: Example of a scan and its visualization in Rviz

As what happens in many real cases, the direction of the rays corresponding to an angle of 0 degrees is parallel with the floor plane, but unlike what happens with the real PIRs, where the vertical field of view grows downwards and decreases upwards, in the simulated model the angle of the vertical field of view only increases downwards, since the sensor only rotates clockwise. This implementation choice is justified by greater simplicity that derives, and it is completely transparent to the user, which is left to a greater flexibility, setting the parameters, such as a minimum and a maximum angle and the height of the sensor. Another implementation choice, this time dictated by the need to improve the data consistency, is the saw-tooth shape of the sensors movement. Many sensors rotate the laser from the minimum angle to the maximum angle gradually, and vice versa from the maximum to the minimum angle always passing through the intermediate angles. This however does not lead to consistency on the timing level, because each successive scans of the same angle are performed within a time interval that is not the same for all the angles. In the simulated sensor once the scan relative to the maximum angle is performed, the laser is tilted suddenly and moved back to the position relative to the minimum angle. The beams only cover a single line-of-sight, leaving the detection area largely uncovered. Therefore the tilting is necessary, but the number of angles has to be controlled because to every new position of the sensor corresponds a new dimension in the matrix (a new array of as many floating point elements as the number of beams) and operations can slow down the entire execution. Areas between the beams are called dead zones. In simulation we can easily increase the number of beams by increasing the resolution, but it is a double-edged sword because it will increase the rendering job of the rendering engine as well as the matrices size. The placement of the devices is important to prevent false alarms, for instance mounting the PIRs in such a way that the PIR cannot "see" out of a window, in case of a window facing a public sidewalk. Using

a laser rangefinder allows us to avoid this problem, because the window will stop the laser beam and a person moving on the other side of the glass would not be ”seen” by the PID, even in the simulation.
For very low velocity movements such as a movement of a hand, the sensitivity reduces sharply with an increase in the distance. Studies demonstrate that if a detector is used in a room of dimension half the maximum design distance of the detector, there exist dead points where detection sensitivity is very low and may give rise to false alarms [25]. Moreover, movements directly towards or away from the detector may sometimes escape detection as the PIRD tends to be less sensitive to this type of movement (radial movements), and a single detector may fail to respond in the extended range. Using a laser rangefinder is sufficient to monitor the occupancy pattern of the room reliably, especially when it deals with radial movements: object moving towards or away from the detector are detected easier than the movements perpendicular to the rays.



(a) Scan at time t *(b) Scan at time t+1*

Figure 4.5: Example of two consecutive scans within the vertical field of view

Chapter 5

Slotted optical switch

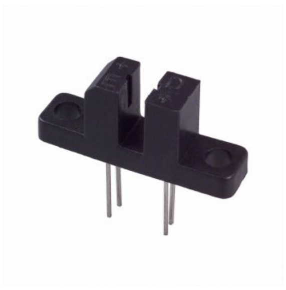


Figure 5.1: Slotted optical switch

The slotted optical switch, sometimes known as opto switch or optical switch (not to be confused with the optical component), is a device comprising a photoemitter (e.g. LED) and a photodetector (e.g. photodiode) mounted in a single package so that the photoemitter normally illuminates the photodetector, but an opaque object can be inserted in a slot between them to break the beam. Slotted switches are often used to detect motor speed by placing a slotted wheel on the motor shaft; as the shaft rotates, it alternately blocks and unblocks the light path [9]. In Ambient Assisted Living applications and in particular in the simulated experimental testbed scenarios, optical switches are used as indicators when a door or hood is open or closed, monitoring the state of the doors or the shutters and the open/close events. This is implemented not only to detect if an elder person left the fridge open, but can be extended to study the daily activities of people living there in a non-intrusively way.

5.1 How the real sensor works

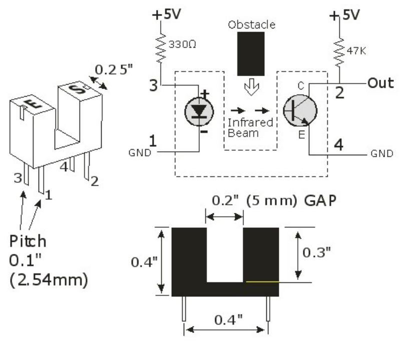


Figure 5.2: Mechanical package information

The slotted optical switches consist of an infrared emitting diode (e.g. LED) facing a photodetector (e.g. photodiode) in a molded plastic housing. A gap separates the two, so if something moves within the gap, it blocks the light path between the LED and the photodetector. A slot in the housing between the emitter and the detector provides a means of interrupting the signal [7].

5.2 How the simulated sensor works

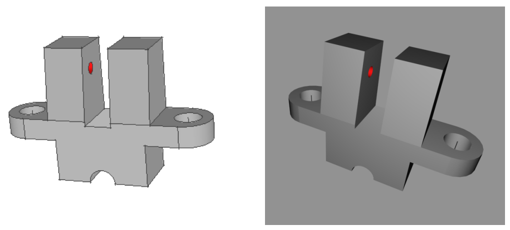
A typical use for slotted switches is as indicators when a door or hood is open or closed. When the door is closed, a flag attached on the door frame drops into the slot and blocks the light.

5.2.1 Design

The simulated model consists of two links and one joint. The links represent both the emitter and the detector elements (components) connected by a fixed joint. The following code shows an example:

```
<sensor name='optical_switch' type='ray' always_on='1'
        update_rate='1' visualize='false'>
  <ray>
    <scan>
      <horizontal samples="1" resolution="1"
                  min_angle="-0" max_angle="0" />
    </scan>
    <range min="0.001" max="0.5" />
  </ray>
  <plugin name="angen_optical_switch"
          filename="libangen_optical_switch.so">
    <alwaysOn>true</alwaysOn>
    <gaussianNoise>0</gaussianNoise>
    <hokuyoMinIntensity>101</hokuyoMinIntensity>
    <updateRate>1.0</updateRate>
    <topicName>door_state</topicName>
        <frameName>link1</frameName>
  </plugin>
</sensor>
```

The link elements are modelled as thin parallelepipeds and to make it more real the visual shape is described by a trimesh, scaled of a proper scale. The collision properties are a bit different to the visual properties of the model. Since the laser is within the emitter element bounding box, the laser ray that starts inside the element would hit the bounding box measuring the same value and in this way nullifying the purpose. The plugin, in order to work correctly, needs a ray sensor as its parent, and since in Gazebo the sensor lacks a physical representation, to gain embodiment it has to be incorporated into the model. The sensor element describes the basic properties of the visual sensor, such as the name, the type and the frequency at which the sensor data is generated. Other attributes contribute to characterize the sensor, like the pose of the sensor optical frame, relative to the sensor parent reference frame, the horizontal samples, and the number of simulated rays to generate per complete laser sweep cycle and their resolution. Lastly the angles of the first and the last range measurement in radians and the smallest and the largest distances that can be measured by the currently connected device. One single ray is sufficient to obtain a correct estimate but in order to have a precise measurement the resolution of the ray has to be quite high since the distances to measure are small and the step size may become bigger than the threshold. The choice of the maximum and minimum range is quite flexible, on condition that the second is smaller than the threshold. The angles have to be set to 0 to make the laser perpendicular to the link. The laser is attached on a link and is fixed. That means that it is sufficient to move only the link and the laser will orientate according to it. In simulation, large sensors will slow down overall performance. Depending on update rates required, it is once again recommended to keep the ray resolution and update rates as low as possible. The plugin itself is characterized by two parameters: the name of the library generated from the compiled source file and the update rate at which the plugin is throttled, that should be the same of the sensor. For simplicity and bigger precision the gaussian noise is removed (set to zero). Another important attribute, common to all the sensor plugins, is the name of the topic the sensor will send data on.



(a) Sketch-up format *(b) Collada file*

Figure 5.3: Three-dimensional model of the optical switch

5.2.2 Implementation

The optical switches are used in the Ängen testbed apartment to monitor the state of many actuators as doors and shutters. The implementation of these sensors is quite simple and it is based on a ray sensor (once again a laser range-finder as for the motion detector). The key concept is to have a ray between two links situated on the top part of the door frame, and a small card on the top of the door or the shutter. When the door is closed, the small card is situated in between the two links and the ray is interrupted. In that way the range value measured by the range finder is minor of a chosen threshold and the closing event is triggered. In order to do that we have to get the laser data and retrieve the range values (one in this case). The message the ray sensor sends out in the interface is the same showed before for the motion detector:

```
std_msgs/Header header
float32 angle_min
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan_time
float32 range_min
float32 range_max
```

```
    float32[] ranges
    float32[] intensities
```

The field we are interested in is *ranges*, an array of floats that specifies the distance of the first object on the ray path that breaks the ray, while the message published by the plugin is:

```
    std_msgs/Header header
    bool open
```

where *header* contains the current time stamp and *open* contains the boolean value that specifies the door state, whether the door is open(true) or closed(false).

An example of the output:

```
header:
    seq: 0
    stamp:
        secs: 306
        nsecs: 121000000
    frame_id: /link1
open: True
```

A difference with the pir sensor seen previously is that the optical switch is memoryless. While the range values of the pir must be compared with the previous ones stored in the matrix, the switch range value is compared with a fixed threshold every time, so that its operation is not affected by the history of the preceding data.

5.3 Example

Optical switches have initially been tested in a empty world, and only at a later stage they have been integrated with the environment. To start the simulation two ways are available, depending if the world is already started, it is possible to spawn the single sensor by command line or through a Gazebo launch file.
Once the sensor is spawned, it will not be active from the beginning. Only when some node will start to listen to the topic subscribed by the sensor, the latter will start the execution. This choice is driven by the search for optimization of the computational resources and the try of lowering as much as possible the impact that every sensor has on the simulator performances.

As explained in the following ”Benchmarks” section, the use of some parameters rather than others and the addition of a sensor will cause a severe speed reduction, sometimes making the simulation even slower than the real time. This plugin implementation assume that every door or shutter has a small flag on it to interrupt the laser ray. Although at first glance it might seem more immediate to utilize more than one ray in order to have a more robust detection, the use of a single ray has proved to be more effective, even as a matter of consistence: if the flag that interrupts the light path only partially breaks the beams stream, interrupting only some of the rays, it could leave the device halfway on, causing an ambiguous output, and this requires a computation overhead for the disambiguation.

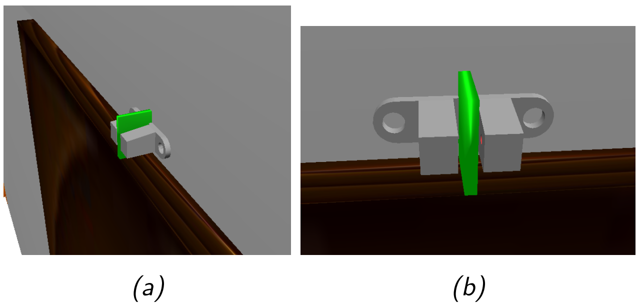


Figure 5.4: The sensor spawned in the simulated environment on the door frame. From the figure is also possible to see clearly the card (the green rectangle) put on the top of the door used to break the ray.

Chapter 6

Pressure sensor

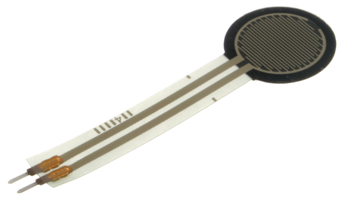


Figure 6.1: Interlink Electronics FSR-402 Force Sensing Resistor.

Pressure sensors are used for control and monitoring in thousands of everyday applications, providing robust solutions for the appliance, medical, consumer, industrial and automotive markets. The measurement of interest is pressure, stated in terms of force per unit area. A pressure sensor usually acts as a transducer, generating a signal as a function of the pressure imposed. Pressure sensors can vary drastically in technology, design, performance, application suitability and cost. The modern pressure sensors are ultra-small and thin, made of silicon, and provide extremely high resolution measurements thanks to the use of the innovative MEMS technology. A conservative estimate would be that there may be over 50 technologies and at least 300 companies making pressure sensors worldwide [8]. In Ambient Assisted Living pressure sensors are the core of the smart home ecology and an important tool for a daily activity recognition system for elder people. The proposed system in the Ängen facility installs pressure sensor to furnitures and floors, to monitor the elder people living there, react in the emergency cases or simply recognize daily activities based on the object usage information. In this case the smart environment can provide a warning sign for unhealthy cases such as skipping meals. For example, to recognize a meal activity, pressure sensors can be installed to each table or chair.

For sleep recognition we can install pressure sensors to a bed in order to capture a lying posture. Pressure sensors can be also installed to sofa and toilet bowl to recognize rest and excretion activities, respectively. Eldercare has been studied for long time and many previous works adopted systems based on RFID or CCTV, where cameras were installed in home and the captured images analyzed. Pressure sensors come forward as a good compromise, resolving the problems of wearing a big and heavy reader in case of RFID-based method and invasion of privacy in case of CCTV-based method, obtaining objects usage information noninvasively.

6.1 How the real sensor works

As a pressure sensor we use a force sensing resistor (FSR), as it is shown in Figure 6.1. FSRs are sensors that detect physical pressure, squeezing and weight. They are simple to use and low cost. Force-sensing resistors are two-wire devices, consisting of a robust piezoresistive conductive polymer thick film, with a variable resistance as a function of applied pressure to its surface. In this sense, the term force-sensitive is misleading: a more appropriate one would be pressure-sensitive, since the sensor's output is dependent on the area on the sensor's surface to which force is applied. These devices are fabricated with elastic material in four layers, consisting of:

- A layer of electrically insulating plastic;

- An active area consisting of a pattern of conductors, which is connected to the leads on the tail to be charged with an electrical voltage;

- A plastic spacer, which includes an opening aligned with the active area, as well as an air vent through the tail;

- A flexible substrate coated with a thick polymer conductive film, aligned with the active area. This polymer is very often replaced by a layer of FSR ink.

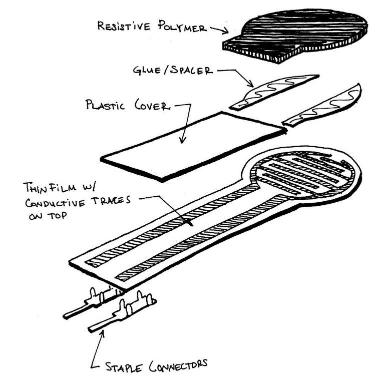


Figure 6.2: FSR parts description

When external force is applied to the sensor, the resistive element is deformed against the substrate. Air from the spacer opening is pushed through the air vent in the tail, and the conductive material on the substrate comes into contact with parts of the active area. The more of the active area that touches the conductive element, the lower the resistance [1]. When there is no pressure, the sensor looks like an infinite resistor (open circuit), as the pressure increases, the resistance goes down (the resistance is inversely proportional to the force applied), as shown in Figure 6.3.

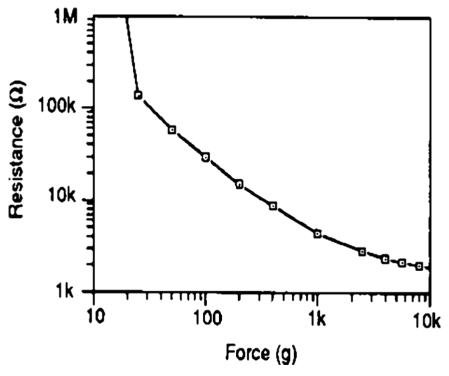


Figure 6.3: FSR characteristic curve. Note that the graph isn't really linear but it is a log/log graph.

Force-sensing resistors are normally supplied as a polymer sheet or ink that can be applied by screen printing. The sensing film consists of both electrically conducting and non-conducting particles suspended in matrix. The particles are sub-micrometer sizes, and are formulated to reduce the temperature dependence, improve mechanical properties and increase surface durability. Applying a force to the surface of a the sensing film causes particles to touch the conducting electrodes, changing the resistance of the film. As with all resistive based sensors, force-sensing resistors require a relatively simple interface and can operate satisfactorily in moderately hostile environments [13]. Compared to other force sensors, the advantages of FSRs are their size (with a thickness typically less than 0.5 mm), low cost and good shock resistance. However, FSRs will be damaged if pressure is applied for a longer time period (hours). A disadvantage is their low precision: measurement results may differ 10% and more.

6.2 How the simulated sensor works

A pressure sensor usually acts as a transducer, it generates a signal as a function of the pressure imposed, expressed as a force per unit area. The sensor used in the implementation however is a contact sensor, a sensor that detects collisions between two object and reports the location of the contact associated forces. The force in question is the ground reaction force. In physics the ground reaction force is the force exerted by the ground on a body in contact with it. For example, a person standing motionless on the ground exerts a contact force on it (equal to the person's weight) and at the same time an equal and opposite ground reaction force is exerted by the ground on the person. What the sensor does is to retrieve the component of the total ground reaction force on the z axis exerting on it.

6.2.1 Design

The model of the pressure sensor used in simulation is very simple. It consists in fact of only one link, with no joints. The code below shows an example of the plugin:

```
<sensor name='pressure_sensor' type='contact' always_on='1'
        update_rate='1' visualize='0'>
  <contact>
    <collision name="link_geom">link_geom</collision>
  </contact>
  <plugin name="angen_pressure_sensor"
          filename="libcontact.so">
    <alwaysOn>true</alwaysOn>
    <updateRate>1.0</updateRate>
    <bumperTopicName>pressure</bumperTopicName>
    <referenceFrameName>floor</referenceFrameName>
    <frameName>link</frameName>
  </plugin>
</sensor>
```

The collision element is a thin box but with a huge mass. This value is so high because the physics engine of the simulator accentuates the collisions between the objects and for some reasons a link with a small amount of mass might go unstable and consequently explode. A solution may be reducing the physics solver timestep, but the expedient of raising the mass of the sensor allowed to keep the physics engine unchanged. The visual element is instead once again a trimesh, opportunely scaled.
The plugin, in order to work correctly, needs a contact sensor as its parent. The contact sensor is attached to the link within the box model. It will report collisions between the box collision object and any other object in the world. One important parameter is the collision name of the contact tag. Since multiple instances of collision tags can exist for the same link (in that case the union of the geometry they define will form the collision representation of the link), it is necessary to specify which is the correct collision object we need to associate the contact sensor to. Another important parameter is the reference frame name. This parameter is required in order to retrieve the correct ground reaction force. Since the pressure sensor is implemented as a contact sensor, all the forces exerting on it will be measured. The reference frame name parameter will tell the controller to measure only the forces exerting on the sensor from that frame (e.g the floor) exclusively.

6.2.2 Implementation

As said before, the sensor retrieves the component of the total ground reaction force on the z axis exerting on it. Among all the forces retrieved by the contact sensor only the forces exerting on the reference frame are considered. The value of the pressure is calculated summing up all the z-axis components of the forces, and subtracting the opposite force due to the sensor weight.
The message published by the contact sensor contains a set of states:

```
Header header
gazebo_msgs/ContactState[] states
```

where every state is composed by the following information:

```
string info
string collision1_name
string collision2_name
geometry_msgs/Wrench[] wrenches
geometry_msgs/Wrench total_wrench
geometry_msgs/Vector3[] contact_positions
geometry_msgs/Vector3[] contact_normals
float64[] depths
```

The information we are insterested in is contained in the field *total_wrench*, that stores the sum of both the forces and torques in every degree of freedom.
An example of the output:

```
header:
  seq: 0
  stamp:
    secs: 829
    nsecs: 269000000
  frame_id: /world
data: 0.0434107481423
```

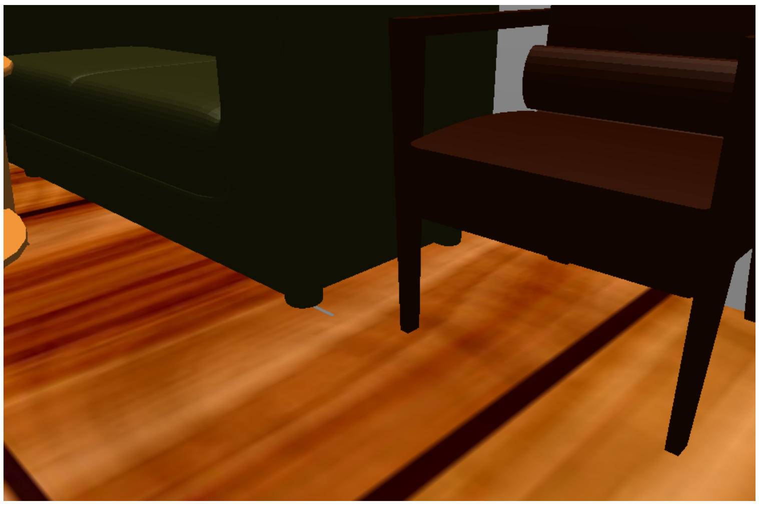
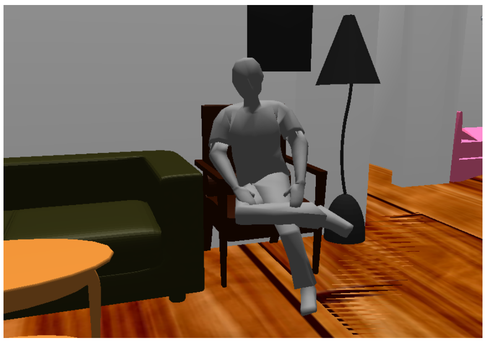


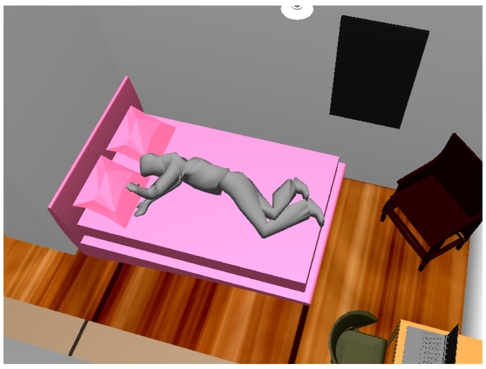
Figure 6.4: Pressure sensor installed underneath a sofa

6.3 Example

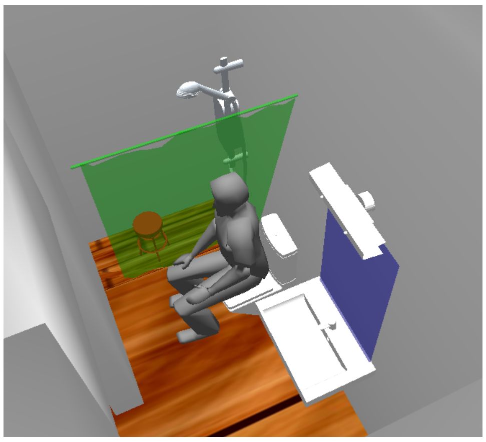
In order to test the pressure sensor system, we install pressure sensors to furniture and floors of the virtual home. Even though the real sensors are fairly low cost and easy to use, they are rarely accurate. Therefore only ranges of response can be expected, and even if they can detect weight, they're a bad choice for detecting exactly how many kilograms of weight are on them. The initial objective was to effectively detect only the presence or the absence of a person on a chair or a sofa and their accuracy indicated a performance of the sensor ON/OFF states correctness, when an old person gave a force to furniture or floors during sitting and lying activities. Subsequently it is thought also to the recognition of objects, linking each of them to a weight and comparing the force exerted by the object on the sensor. In simulation this is possible by simply observing the contacts detected by the bumper and reading in the message published by the bumper sensor the name of the collision box that collides with our device's collision box. With a proper nomenclature of the objects their recognition is immediate. Especially with the pressure sensors the physical engine simulator is very realistic: for instance if the sensor is placed under the leg of a table and an object is moved on the table, the weight detected by the sensor varies according to the position, it will be greater as it is near the corner of the table corresponding to the foot beneath which is the sensor, smaller as it is away from it. Even putting objects on top of others, the result is the same, as the weight is exerted transitively on the foot of the table. Another disadvantage is that the information about the name of the object on top is lost, as the bumper only detects the collision with the object that actually touches it, even though the total weight is still the sum of all the 'stacked' objects. By setting the weight of the human model appropriately and retrieving the contact positions returned by the sensor, even a different position or posture and therefore a different activity executed by the user can be detected in a consistent way from the pressure sensor.



(a)



(b)



(c)

Figure 6.5: Example of different scenarios: a human seated on a chair (a), lying on the bed (b) and on the toilet (c)

Chapter 7

Light sensor

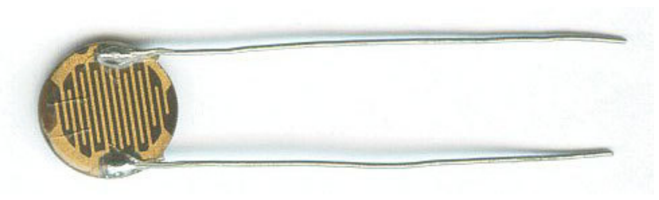


Figure 7.1: Light Dependent Resistor (LDR)

Light sensing devices are used in almost any branch of industry for control, safety and amusement. From cell phones where they enable automatic control of display backlight brightness to automobiles where the use of LCDs is increasing for navigation, entertainment and comfort systems as well as control monitoring and dimming mirrors. In Ambient Assisted Living light sensors are used for automatic room light detection and control, in order to lower the home power consumption and save energy. Sometimes in fact, the light intensity from outside is sufficient and we don‘t need to turn on any light, but often the user leaves the home forgetting to turn off the light and this causes a waste of energy. In the testbed facility apartment, light sensors are installed on every lamp. By using the PIR sensor seen before, the light control system can detect if a human body enters the detection area or not. In case there is no human body present, all controlled lights are turned off. If there is, the sensor detects the light intensity under the environment and maintains sufficient light by controlling the number of lights. Today designers have more technology choices for ambient light sensors including photoelectric cells, photodiodes, phototransistors, and photo ICs. In the real Ängen facility, light sensors use a photoresistor to detect the light intensity of the environment. A photoresistor or light dependent resistor (LDR) is a resistor whose resistance decreases with increasing incident light intensity; in other words, it exhibits photoconductivity. There are many types of photoresistors, with different specifications and models. Photoresistors can be coated with or packaged in different materials that vary the resistance, depending on the use. Many of them are radiometric and characterize the distribution of the radiation's power in space, many are photometric and characterize the light's interaction with the human eye. Some of them are analog, some others are digital. Some devices contain both infrared and full spectrum diodes, allowing to separately measure infrared, full-spectrum or human-visible light [28]. The SI unit of illuminance and luminous emittance is called lux, which measures luminous flux per unit area. It is equal to one lumen per square meter and it is used in photometry as a measure of the intensity, as perceived by the human eye, of light that hits or passes through a surface.

7.1 How the real sensor works

A photoresistor is made of a high resistance semiconductor. The resistance of the sensor decreases when the light intensity of the environment increases. If light falling on the device is of high enough frequency, photons absorbed by the semiconductor give bound electrons enough energy to jump into the conduction band. The free electron (and its hole partner) conduct electricity, thereby lowering resistance. The amount of light detected can be read as an analog value or set to trigger a digital signal when a threshold is reached [21].

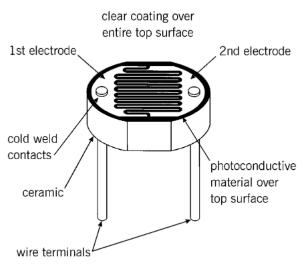

clear coating over entire top surface
1st electrode
2nd electrode
cold weld contacts
ceramic
photoconductive material over top surface
wire terminals

Figure 7.2: Sensor description

7.2 How the simulated sensor works

The aim of the simulation is to abstract the physical architecture and simplify the operating principle. Previous attempts tried to obtain the material properties of the objects spawned into the world, trying to retrieve the specular and reflective components of the light incident on them, or access the lights themselves to deduce the contribution of each of them, but that was not possible at run-time because the simulator had no interfaces of that kind exposed. One idea was to access the main camera and look for some parameter that could refer to the luminosity of the entire scene, but that feature is implemented only within the stand-alone version of Gazebo. One solution seemed to be dealing directly with the rendering engine, since it needs the light information to draw the scene, but once again, it was not possible to act in such a low level cause the simulator developers built the rendering engine strongly decoupled from the simulator itself. The solution implemented takes the Lego NTX Light Sensor [19] as its starting point. This device enables to distinguish not only between light and dark, but can only determine the light intensity of different colors.

7.2.1 Design

As already mentioned, one of the advantages of the simulation is the abstraction of all the physical and architectural details. The model of the simulated light sensor implemented in the environment is composed of a single link, with cylindric shape, and no joints. The camera sensor is required as the parent of the plugin. In order to work properly the image format has to be set to RBG (R8G8B8). For simplicity the image size is reduced to one pixel only. The light spread out uniformly on the walls of the simulated apartment model so the result would be the same even using a bigger image size. Another essential parameter is the farClip parameter. It is important that the ray of the camera is able to reach the wall to sample the pixel values. Introducing a distance that is not big enough will cause an incorrect measurement. The code below shows an example of the plugin:

```
<sensor name='light_sensor' type='camera' always_on='1'
        update_rate='1' visualize='false'>
  <camera>
    <horizontal_fov angle='0.02' />
    <image width='1' height='1' format='R8G8B8' />
    <clip near='0.5' far='5' />
  </camera>
  <plugin name="angen_light_sensor"
          filename="libangen_light_sensor.so">
    <alwaysOn>true</alwaysOn>
    <imageTopicName>image</imageTopicName>
    <luminosityTopicName>luminosity</luminosityTopicName>
    <updateRate>1.0</updateRate>
    <cameraName>light_sensor</cameraName>
    <frameName>link</frameName>
  </plugin>
</sensor>
```

7.2.2 Implementation

Photosensors or photodetectors are sensors of light or other electromagnetic energy. The operating principle is similar to the principle used by a particular photosensor called image sensor, a device that converts an optical image into an electronic signal. What the sensor does is to use a camera to retrieve the RGB values of four pixels, each one taken by each wall of a room. This method works well because the color of the walls is white. This allows the pixels to vary between absolute white (255) and absolute black (0), while other colors spaced in a smaller range, making the calculation harder and less precise. Once the values are retrieved, the luminance is calculated. Luminance is a photometric measure of the luminous intensity per unit area of light travelling in a given direction. For RGB color spaces luminance can be calculated as the linear combination of the standard chromaticities of red, green, and blue with the formula:

$$Y = 0.2126 * R + 0.7152 * G + 0.0722 * B$$

Since a camera sensor is used, we will benefit from the camera plugin already implemented in ROS to retrieve the image, one pixel in our case. The message published by the camera controller is the following:

```
Header header
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step
uint8[] data
```

The only piece of information we are interested in is stored in the array of integers that represents the actual matrix data. Since we have only one pixel and the image is in the RGB format, the array will contain three integer values, that specify respectively the red, green and blue channel components. Once these values are extracted the luminance is calculated with the formula seen before and published through the LightSens.msg message:

```
    std_msgs/Header header
    float32 data
```

where *data* represents the value averaged out from the four pixel values and reproduced on a percentage scale. An example of the output:

```
header:
    seq: 0
    stamp:
        secs: 5632
        nsecs: 333000000
    frame_id: /link1
data: 11.37254905723
```

7.3 Example

The addition of furniture in the empty world for the simulation of the light sensors does not affect their operating principle, unless the furniture are placed in between the camera and the walls. As explained before, the camera retrieves the pixels of the walls since they are white and this allows to calculate a consistent value. To avoid this problem and not to limit the flexibility in placing the furniture, the sensors are positioned high up in the middle of the rooms. At the beginning it was thought to point the camera on the floor, but its colour does not vary uniformly, remaining almost stable to the same RGB values. Even by positioning the sensor to make it pointing the light does not solve the issue and the values are not coherent since lights in Gazebo are not conceived as visual objects out-and-out. To retrieve the wall pixel colour has been proved to be the best option. The choice to use more than one pixel is due to the coherence of the model: by pointing the camera on a single wall, the sensor would retrieve an high level of luminosity in case a directional light is pointed on it, even if the entire room would be not as lighten. By averaging the contribution of all the four walls the obtained value is instead consistent. Using four cameras may be computationally expensive and overburden the simulation, so a single camera is rotated four times. Finally the sensor is able to retrieve a wide spectrum, with a range that goes from 0.39%, obtained deleting any light from the world, to 99%, obtained placing two lights in the middle of the room.

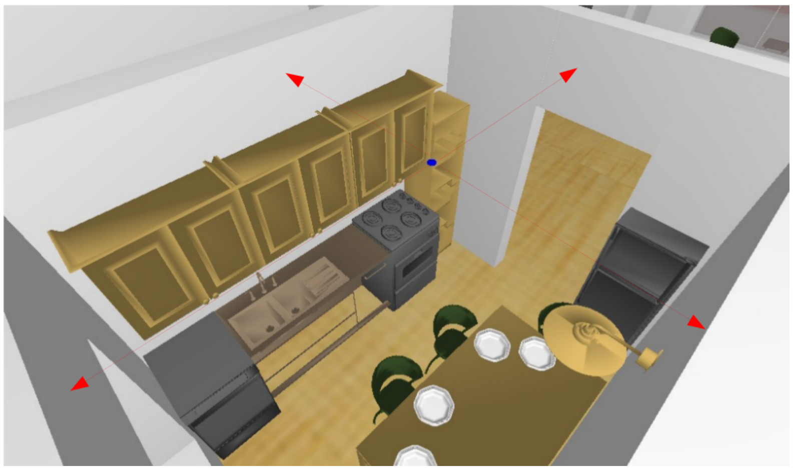


Figure 7.3: The four pixels of the walls sampled in the kitchen by the light sensor.

Chapter 8

Evaluation

Not all the simulations are the same. A criteria to establish which simulation is more effective is without a doubt the evaluation of parameters like computational load, CPU and memory occupancy, speed and reproducibility. A model that requires lots of computational resources and fill up the CPU, slowing the entire simulation, should be discarded.

8.1 Computational efficiency

A previous simplistic approach implemented the sensors decoupling the model from the plugin, moving the logic to an external node. This caused a sensible deterioration of the performances, and a big slowing down of the simulation time. Every single node executed requires in average 10% of the CPU and it produces overhead on the transport layer, due to serialization and deserializazion of the messages used for communication. Considering that the simulator requires up to 60% of the CPU, it may be irrelevant for a test with only few sensors but it becomes inconceivable if we want to simulate a real environments with more than ten sensors, especially if we add the presence of robots. Using plugins is convenient but still many improvements can be obtained with a proper combination of parameter.

One solution is to make the model as simple as possible. Removing for instance the collision component when it is not necessary can reduce computation time. Another approach can be to downsample the sensors as much as possible or to downsample the physics. Even if it is required to increase the number of quick step iterations per physics loop, decreasing the update rate is still an advantage because collision checks are not executed so often. Obviously, this can cause problems in physics quality, and objects/robots can "explode". It is necessary to tune controller gains for a lower update rate. An extreme solution may be to reduce CPU load by disabling the simulator GUI. A large performance hit to the simulator is laser scan generation. The laser data is generated with ODE collision checking, and a very noticeable difference can be seen in speed when subscribing to the laser data. For the motion detector and the optical switches, tests are executed measuring the impact of the number of rays that composed the beam. The result, shown in Figure 8.1, reveals that reducing the number of rays from 640 to 160 for each laser, the simulation is speeded up of a factor of 10X. Another important aspect visible from the plot is that keeping 160 rays is not so much more heavy than having 80 rays, and we are able to maintain a high level of accuracy, up to 90%.

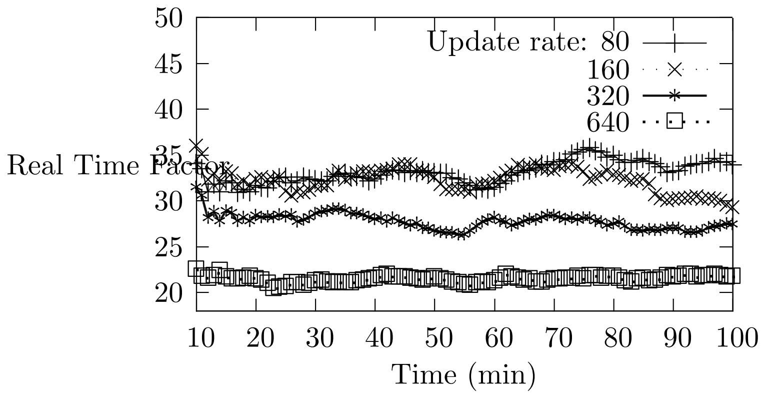


Figure 8.1: Impact on the simulation speed of the number of rays composing the laser beam.

Figure 8.2 shows instead the impact of the number of motion detectors added to the environment. Every time we add a new sensor, the simulation is slowed up of a factor of 5X, a value rather acceptable.
Camera rendering is also expensive. Since we have only one-pixel image to render, the rendering is not expensive. Furthermore the sensors are implemented in a way that if there are no nodes subscribing to the camera topic, the data will not be generated. Another choice imposed by performance requirements has been to not use point clouds. The cost to tilt the laser and execute an arbitrary number of scans is lesser than the cost to elaborate a three-dimensional scan and the quality of the operative principle is not subjected to a worsening. For that concerns optical switches experiments shown that the ray resolution does not affect performance, since the beam is composed by a single ray. The impact on performance of light sensors is tested varying the distance of the far plane. Increasing for example the

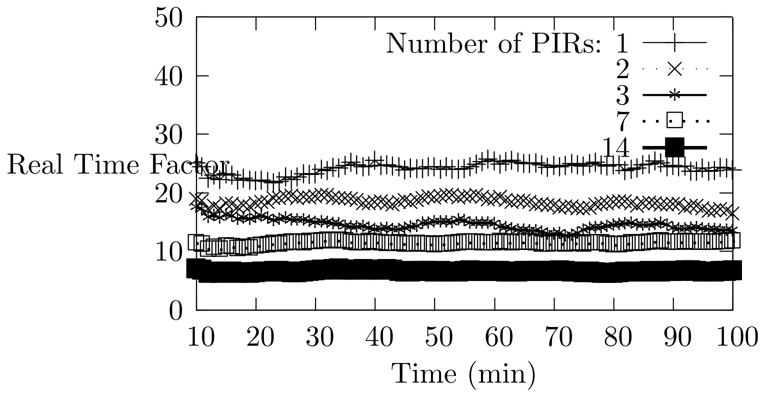


Figure 8.2: Impact on the simulation speed of the number of PIRs.

range from 10 meters to 25, we assist to a slow down of 1-2%. It is a compromise if we want to measure for instance the light in a wide surrounding like a spacious living room or a warehouse.

8.2 Integrated System

All the experiments have been done on a Intel(R) Core(TM) i5-2435M CPU @ 2.40GHz machine. The whole ecology of sensors implemented in the test environment is composed by 22 sensors: 7 motion detectors, 7 optical switches, 7 light sensors and 1 pressure sensor.

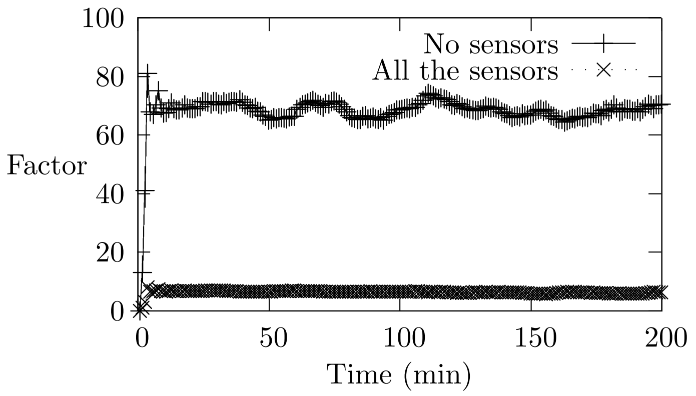


Figure 8.3: Simulation performances with all the sensors active.

As shown in Figure 8.3, integrating the environment with the sensors, the simulation time is slowed down from a factor of 70X to a factor of 10X. The biggest contribution to the computational load is given by the light sensor, as it makes use of a camera, and by the motion detector, as it processes relatively wide matrices of data every second.

Chapter 9

Portability

As almost all the open source (and relatively recent) software, supported by a very active and fertile community of users and developers, ROS and consequently Gazebo have been subject to changes, both bugfixes and improvements, that led to version updates, often very close to each other. Now Gazebo is a stand alone project and users are highly discouraged from using the documentation and tutorials for Gazebo on the ROS wiki page. Below are listed the main changes.

9.1 From Electric to Fuerte

Previous versions of Gazebo (including Electric) utilized controllers. These behaved in much the same way as plugins, but were statically compiled into Gazebo. Plugins are more flexible, and allow users to pick and choose what functionality to include in their simulations. Furthermore to update from gazebo's pre-1.0.0 release to the latest revision a plugin conversion has been necessary. The changelist highlights of that API update concern files including, class inheritance (no longer a controller but a specific type, *SensorPlugin*), constructors, loading and initialization functions, parameter parsing and registration macros.

9.2 Troubleshooting in Fuerte

With Fuerte there have been introduced some bugfixes concerning some sensor implementations, from cameras to ray sensors, up to the contact bumper. While the first improvements did not influence the work since they dealt with z-buffering or image rendering of point clouds, which we were not interested in, the bug that came together with the contact sensor gave us a hard time. Until the bumper was spawned into the world, everything worked as expected and the plugin was loaded, but when we tried to listen to the topic an error popped up on the console saying that the sensor did not have any collisions. The problem was that the first of two piped functions returned only the name of the collision object/geometry while the second one used the scoped name to fill the contacts map. The solution was to replace a method call to retrieve the scoped name this time, modifying a single line of code. What made things difficult was to modify the source file that could not be accessed on the online Gazebo repository. The only way to apply the patch was to create a stack overlay with the latest tag which contained the bugfix.

9.3 From Fuerte to Gazebo stand-alone

As mentioned before, Gazebo is now a stand alone project. The previous version of Gazebo was a monolithic implementation where the user accessed to sensors and controllers through a shared memory interface. These implementation caused issues in relation to the speed and flexibility of the simulator. In order to address the requirements and resolve the current issues, a new distributed architecture was proposed. Gazebo has been broken into libraries for physics simulation, rendering, user interface, communication, and sensor generation, providing three different processes that communicate using a combination of google::protobufs and sockets. The stand alone version also defines a new format to load and save information about a simulation world or model, called Simulation Description Format (SDF). The standardized URDF format used in ROS is lacking many features and has not been updated to deal with the evolving needs of robotics. URDF can only specify the kinematic and dynamic properties of a single robot in isolation but it cannot specify the pose of the robot itself within a world, and it is also not a universal description format since it cannot specify joint loops (parallel linkages), and it lacks friction and other properties. Additionally, it cannot specify things that are not robots, such as lights, heightmaps, etc. On the implementation side, the URDF syntax breaks proper formatting with heavy use of XML attributes, which in turn makes URDF more inflexible. There is also no mechanism for backward compatibility. On the other hand, SDF is a complete description for everything from the world level down to the robot level. It is scalable, and makes it easy to add and modify elements. The SDF format is itself described using XML, which facilitates a simple upgrade tool to migrate old versions to new versions. It is also easily extensible and self-descriptive. Many other improvements are done concerning speed (decoupling physics, sensor generation, and gui into separate processes which can be run independently), modularity (the user can pick and choose which processes to run and where to run them), and portability (the user can run complete simulation on a cluster and run the graphic interface on local machine). Other important features can be listed below:

- access to multiple physics engines including ODE and Bullet
- direct control to all aspects of the simulation engine including the phyics engine, graphics libraries, and sensor generation
- possibility to run Gazebo on remote servers
- new tool to construct a 3D model of a building within the Gazebo UI, without writing a line of code
- redesigned graphical interface that provides access to simulation properties, modification of models, and drag-and-drop insertion of models
- new sensors (e.g. RFID) and robots provided
- human simulation: replay of human motion capture data in a running simulation.

Chapter 10

Conclusions

The population of elderly people and individuals with disabilities is in constant need of continuing research in order to improve their independent life and their possibility to age in place. With the smart home technology the needs for those people can be met and relive the workload from family and health providers. However successful integration of supporting intelligent systems into everyday life depends on end users acceptance of the technology and how it affects their daily life. A good compromise of ubiquity, reliability and non-intrusiveness is established by the ambient sensors. By rapid prototyping these sensors a significant reduction in efforts and complexity is obtained, and using proper development and simulation tools such as the Robot Operating System (ROS) and its 3D simulator Gazebo shown before allows to maintain a high degree of flexibility and possibilities for reuse.

10.1 Achieved objectives and future work

The implemented plugin allows to create the ambient sensors with the possibility to set many of the parameters that characterize the real counterparts and that can be found in almost all the datasheets. Every sensor is a standalone device that can be spawned anywhere in the world and moved as the user prefers. Due to its modular architecture, the entire ecology of sensors can be easily extended by additional modules to provide other functionalities. It is possible to create a launch file and group together the sensors within the environment to start the simulation together with other actuators or robots, or individually through the command line if the simulation is already started. The implementation maintains the simulation performance quite high. The sensors are not active as soon as they are created, even if the

simulation is running. Only when the topic of the sensor is subscribed by some other node, it starts to produce data and fill the customized messages that will be published on its own topic. These sensors can be connected to some distributed computing system and send the information collected from the simulated environment to all the actuators and robot present in the surroundings, in order to perform a proactive action consequently to a state change or anomalous event.
Future research will be directed towards the implementation of a graphic interface that allows the user to choose the sensor and set the parameters in a more intuitive way, in a form of a small separate application. Another aspect that will be taken in consideration will be the simulation of humans to allow the investigation of interaction-related issues.

Bibliography

[1] Force-sensitive resistor (fsr). http://www.sensorwiki.org/doku.php/sensors/force-sensitive_resistor?s[]=fsr.

[2] Google 3dwarehouse website. http://sketchup.google.com/3dwarehouse/.

[3] Google sketchup website. http://www.google.com/intl/en/sketchup/3dwh/.

[4] Open asset import library. http://assimp.sourceforge.net/.

[5] Open dynamics engine. http://www.ode.org/.

[6] Open-source graphics rendering engine. http://www.ogre3d.org/.

[7] Optical switch datasheet. http://www.optekinc.com/datasheets/opb960_990_series.pdf.

[8] Pressure sensor. http://en.wikipedia.org/wiki/Pressure_sensor.

[9] S. Ball. *Analog Interfacing to Embedded Microprocessor Systems*. Embedded technology series. Elsevier Science, 2003.

[10] S. Ball. Exploring optical and magnetic sensors. *Embedded Systems Programming*, 16(7):16–23, 2003.

[11] ROS community. Ros website. http://www.ros.org/.

[12] D. J. Cook, J. C. Augusto, and V. R. Jakkula. Ambient intelligence: Technologies, applications, and opportunities. *Pervasive and Mobile Computing*, 5(4):277–298, 2009.

[13] R. M. Crowder. Tactile sensing. *Automation and Robotics*, pages 1–5, 1998.

[14] S. Hoberg, L. Schmidt, A. Hoffmann, M. Söllner, J. M. Leimeister, C. Voigtmann, K. David, J. Zirfas, and A. Roßnagel. Socially acceptable design of a ubiquitous system for monitoring elderly family members. In *GI-Jahrestagung*, pages 349–363, 2012.

[15] A. Howard, N. Koenig, J. Hsu, and M. Dolha. Gazebo simulator website. `http://gazebosim.org/`.

[16] A. Kaushik, N. Lovell, and B. Celler. Evaluation of pir detector characteristics for monitoring occupancy patterns of elderly people living alone at home. In *Engineering in Medicine and Biology Society, 2007. EMBS 2007. 29th Annual International Conference of the IEEE*, pages 3802–3805. IEEE, 2007.

[17] H. J. Keller. Advanced passive infrared presence detectors as key elements in integrated security and building automation systems. In *Security Technology, 1993. Security Technology, Proceedings. Institute of Electrical and Electronics Engineers 1993 International Carnahan Conference on*, pages 75–77. IEEE, 1993.

[18] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, 3:2149–2154, September 2004.

[19] LEGO Mindstorms. Light sensor. `http://shop.lego.com/en-US/Light-Sensor-9844`.

[20] M. Philipose, K. P. Fishkin, M. Perkowitz, D. J. Patterson, D. Fox, H. A. Kautz, and D. Hähnel. Inferring activities from interactions with objects. *IEEE Pervasive Computing*, 3(4):50–57, 2004.

[21] I. Poole. Light dependent resistor, photo resistor, or photocell. `http://www.radio-electronics.com/info/data/resistor/ldr/light_dependent_resistor.php`.

[22] M. Quigley, E. Berger, A. Y. Ng, et al. Stair: Hardware and software architecture. In *AAAI 2007 Robotics Workshop, Vancouver, BC*, pages 31–37, 2007.

[23] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[24] Steve's Lock & Safe. Venice locksmith home security technicians notebook. http://shop.lego.com/en-US/Light-Sensor-9844.

[25] B. Song, H. Choi, and H. S. Lee. Surveillance tracking system using passive infrared motion sensors in wireless sensor network. In Y. Park and Y. Choi, editors, *ICOIN*, pages 1–5. IEEE, 2008.

[26] A. Steventon, M. Bardsley, J. Billings, J. Dixon, H. Doll, S. Hirani, M. Cartwright, L. Rixon, M. Knapp, C. Henderson, et al. Effect of telehealth on use of secondary care and mortality: findings from the whole system demonstrator cluster randomised trial. *BMJ: British Medical Journal*, 344, 2012.

[27] WikID The Industrial Design Engineering wiki. Ambient sensors. http://wikid.eu/index.php/Ambient_sensors.

[28] G.H.Y. Ting and J. Woo. Elder care: is legislation of family responsibility the solution? *Asian Journal of Gerontology & Geriatrics*, 4(2):72–5, 2009.

[29] Robotic Design System VEX. Light sensor. http://www.vexforum.com/wiki/index.php/Light_Sensor.