André Gradil

Development of a Remote 3D Visualisation, Control and Simulation Framework for a Robotic Head

Dissertação de Mestrado em Engenharia Eletrotécnica e de Computadores
09/2016

• U C •

Universidade de Coimbra

FCTUC

UNIVERSITY OF COIMBRA
FACULTY OF SCIENCES AND TECHNOLOGY
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Development of a Remote 3D Visualisation, Control and Simulation Framework for a Robotic Head

André de Jesus Gradil

Supervisor:
Prof. Doctor João Filipe de Castro Cardoso Ferreira

Jury:
Prof. Doctor João Filipe de Castro Cardoso Ferreira
Prof. Doctor Jorge Nuno de Almeida e Sousa Almada Lobo
Prof. Doctor Nuno Miguel Mendonça da Silva Gonçalves

Dissertation submitted to the Electrical and Computer Engineering Department of the Faculty of Science and Technology of the University of Coimbra in partial fulfilment of the requirements for the Degree of Master of Science.

Coimbra, September of 2016

Acknowledgements

I would first like to thank my thesis advisor Prof. João Filipe Ferreira of DEEC at University of Coimbra, he consistently allowed this paper to be my own work, but steered me in the right the direction whenever he thought I needed it. My colleagues and friends for having accompanied me through this path both in work and at leisure. Finally, I must express my very profound gratitude to my parents and my brother for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Thank you,
André Gradil

Abstract

In this text, we will present work on the design and development of a ROS-based remote 3D visualisation, control and simulation framework. This architecture has the purpose of extending the usability of a system devised in previous work by this research team during the CASIR project.

Recent advances in robotic development technology will provide the necessary support for an exponential breakthrough in robotic research; in fact, the forecast is that robotics will be witnessing a boom in the near future. Among this technology, development environments such as the Robot Operating System (ROS) framework, simulation tools and the recent trend of remote robotic laboratories will undoubtedly play an important role in promoting this breakthrough. They will most probably lessen the burden on developers and users while they operate the robotic systems they have been entrusted, many times consisting of expensive, cumbersome, diverse equipment supported by complex, intricate software systems.

With this in mind, the proposed solution was implemented using ROS, and designed to be used in three different ways. More specifically it will attend the needs of two user groups – local and remote users and developers. It comprises hardware and simulator access to local users, having an intuitive graphical user interface to support the use of the aforementioned feature, and provides access to remote users through a remote experimental robotic laboratory. Experimentation using the simulated environment allows for the enactment of offline experiments, in order to safely and controllably develop and test new algorithms or components. Remote experimentation will allow researchers outside the laboratory to have access to the CASIR-IMPEP system, in a secure and organised fashion through an interactive website.

To this end, several tools and methods were analysed and compared, from 3D modelling tools such as Blender and Maya, robotic simulator software such as Gazebo and WeBots, several options for user interface software creation, and web solutions to open the Robot Operating System (ROS) to non-ROS machines through a web browser.

Next, the framework was implemented, specifically consisting of: (1) a fully functional simulator integrated with the ROS environment, including a faithful representation of the IMPEP robot with every actuator and sensor emulated virtually, a human model with animation capabilities and enough features for enacting human robot interaction scenarios, and a virtual experimental setup with similar features as the real laboratory workspace; (2) a fully functional and intuitive user interface with 2D and 3D image representation capabilities, also allowing both common and advanced users or developers to launch specific sets of modules; (3) a remote robotic laboratory that can connect remote users to the rest of the framework via a web browser, providing them basic control of the simulated platform, via a virtual joystick controller.

Finally, the proposed solution was thoroughly and systematically tested under operational conditions, so as to assess its qualities in terms of features, ease-of-use and performance. The final outcome was shown to fulfil the overall goals of this work – (1) the development of hardware and simulator access to local users, with support of a intuitive local GUI; (2) providing access to these assets to remote users through a remote robotic lab – and to provide the many features that ensure that a user-friendly, high-performance framework was developed.

Every component acts in a synergistic fashion so as to complement the core CASIR framework in a simple, modular, and optimised fashion, trying in full to minimise resource usage and computational burden of the main computer.

At the end of this dissertation, conclusions concerning the success and potential of this research and development effort are drawn, and the foundations for future work will be proposed.

Keywords: **Visualisation, Simulation, Remote, ROS, Gazebo, Framework, GUI**

"You can mass-produce hardware; you cannot mass-produce software - you cannot mass-produce the human mind." - Michio Kaku

Resumo

Neste texto, vamos apresentar o trabalho sobre a conceção e desenvolvimento de uma arquitetura de visualização remota 3D, controlo e simulação baseada em ROS. Esta arquitetura tem o propósito de estender a usabilidade de um sistema desenvolvido em trabalhos anteriores por esta equipa de investigação durante o projeto CASIR.

Os recentes avanços na tecnologia de desenvolvimento robótico irão fornecer o suporte necessário para um avanço exponencial na pesquisa robótica, na verdade, a previsão é que a robótica vai testemunhar um boom num futuro próximo. Entre estas tecnologias, ambientes de desenvolvimento, tais como a arquitectura "Robot Operating System" (ROS), ferramentas de simulação e a tendência recente de laboratórios robóticos remotos, sem dúvida desempenharão um papel importante na promoção destes avanços. Eles provavelmente irá diminuir a carga sobre os criadores e utilizadores enquanto eles operam os sistemas robóticos lhes foram confiadas, sendo este muitas vezes caro, equipamento pesado ou muito diversificado e apoiada por sistemas complexos de software.

Com isto em mente, a solução proposta foi implementada utilizando ROS, e projetada para ser usada de três maneiras diferentes. Mais especificamente, irá atender às necessidades de dois grupos de utilizadores - utilizadores e criadores locais e remotos.

A solução passa por garantir acesso a hardware e a um simulador aos utilizadores locais, tendo o apoio de uma interface gráfica intuitiva e por fim fornecer acesso a usuários remotos através de um laboratório robótico remoto experimental. Experimentação usando o ambiente simulado permite a recriação de experiências off-line, a fim de desenvolver e testar novos algoritmos ou componentes de forma segura e controlável. Experimentação remota permitirá que pesquisadores externos ao laboratório terem acesso ao sistema CASIR-IMPEP, de uma forma segura e organizada através de um site interativo.

Para este fim, várias ferramentas e métodos foram analisados e comparados, desde ferramentas de modelação 3D como Blender e Maya, software simulação robótica como Gazebo e Webots, várias opções para a criação de software com interface gráfica de utilizador e soluções web para abrir o ROS a máquinas não-ROS através de um navegador web.

De seguida, a estrutura foi implementada, consistindo especificamente em: (1) um simulador totalmente funcional integrado com o meio de ROS, incluindo uma representação fiel do robô IMPEP com todos os actuadores e sensores emulado virtualmente, um modelo humano com capacidades de animação e características suficientes para recriação de cenários de interação do humano/robô, e uma configuração experimental virtual com características semelhantes às do espaço de trabalho real do laboratório; (2) uma interface de utilizador totalmente funcional e intuitiva, com capacidades de representação de imagem 2D e 3D, permitindo que tanto os utilizadores comuns como avançados e até criadores possam correr conjuntos específicos de módulos; (3) um laboratório robótico remoto que pode ligar os utilizadores remotos ao resto da arquitetura através de um navegador web, proporcionando-lhes o controlo básico da plataforma de simulação, através de um controlador joystick virtual.

Finalmente, a solução proposta foi cuidadosamente e sistematicamente testados em condições operacionais, de modo a avaliar as suas qualidades em termos de recursos, facilidade de uso e desempenho. Os resultados finais confirmaram cumprir os objetivos gerais deste trabalho - (1) garantir acesso a hardware e a um simulador aos utilizadores locais, tendo o apoio de uma interface gráfica intuitiva; (2) garantir que os utilizadores remotos tenham acesso a esses recursos através de um laboratório robótico remoto - e fornecer os recursos necessários a fim de garantir a criação de uma arquitetura amigável para o utilizador de alta performance.

Cada componente age de uma forma sinérgica, de modo a complementar a arquitetura central do projecto CASIR de uma forma simples, modular e otimizada, tentando na íntegra para minimizar o uso de recursos e carga computacional do computador principal.

No final desta dissertação, serão retiradas conclusões sobre o sucesso e o potencial deste esforço de pesquisa e desenvolvimento, propondo também as bases para o trabalho futuro.

Palavras chave: **Visualização, Simulação, Remoto, ROS, Gazebo, Arquitectura, Interface**

List of acronyms

CASIR	Coordinated Attention for Social Interaction with Robots.
IMPEP	Integrated Multimodal Perception Experimental Platform.
HRI	Human-Robot Interaction.
ROS	Robot Operative System.
YARP	Yet Another Robot Platform.
IMU	Inertial Measurement Unit.
OSRF	Open-Source Robotics Foundation.
ODE	Open Dynamics Engine.
DART	Dynamic Animation and Robotics Toolkit.
V-Rep	Virtual Robot Experimentation Platform.
GUI	Graphical User Interface.
SDF	Standard Data Format.
STL	STereoLithography.
DXF	Drawing Exchange Format.
MOOS	Mission Oriented Operating Suite.
URBI	Universal Real-Time Behavior. Interface.
API	Application Programming Interface.
GPS	Global Positioning System.

HMI	Human-Machine Interface.
BSD	Berkeley Software Distribution.
LGPL	Lesser General Public License.
HTML	HyperText Markup Language.
JSON	JavaScript Object Notation.
URDF	Unified Robot Description Format.
RGB	Red Green Blue.
HDR	High Dynamic Range.
FOV	Field Of View.
GPU	Graphics Processing Unit.

Table of contents

List of acronyms **xi**

List of figures **xv**

List of tables **xvii**

1 Introduction **1**

1.1 Motivation and overall goals 1

1.2 Related work 2

1.3 Contributions 5

1.4 Structure of dissertation 5

2 Background and methods **6**

2.1 An open-source framework for studying artificial perception and attention in HRI 6

2.2 The Robot Operating System (ROS) 8

2.3 Simulation and 3D modelling tools for human-robot interaction 10

2.4 Graphic user interfaces for robotic applications 13

2.5 Remote robotic experimental laboratories 15

3 Implementation and results **18**

3.1 Putting it all together – a ROS framework for the CASIR-IMPEP platform . 18

3.2 Implementation details for the Gazebo-based simulation package 20

3.2.1 IMPEP simulation – sensors 22

3.2.2 IMPEP simulation – actuators 25

3.2.3 Environmental simulation 27

3.2.4 Avatar and interaction simulation 29

3.3 Implementation details for the *rqt*-based user interface 32

3.4 Implementation details for the web service supporting the CASIR-IMPEP remote lab 36
3.5 Experimental results 39
3.5.1 Simulation and visualisation proof-of-concept 40
3.5.2 Remote simulation and control proof-of-concept 45

4 Conclusions and future work 47

References 50

Appendix A Parsed URDF kinematic chain 53

List of figures

1.1 Desired features for most contemporary robotic projects. 2
1.2 Conceptual diagram for the IMPEP ROS framework for remote 3D visualisation, control and simulation. 4

2.1 Existing Attentional system design. 7
2.2 The Integrated Multimodal Perception Experimental Platform. 7
2.3 CASIR-IMPEP system architecture overview 8
2.4 Simulated cameras framegraber node communication. 9
2.5 From moddeling tools to simulator. 13
2.6 Care-O-Bot3 dashboard (srs_ui_pro) created in wxWidgets. 14
2.7 HMI development frameworks to work with ROS. 15
2.8 PR2 remote laboratory. 17

3.1 Framework goals for the IMPEP platform. 18
3.2 Simplified system dataflow 19
3.3 IMPEP model packages for simulation. 21
3.4 IMPEP virtual model evolution 22
3.5 Demonstration of the effect of Gaussian noise modelling on simulated IMPEP camera image quality 24
3.6 Virtual kinect output. 25
3.7 Revolution joints emulating pan and tilt motors 26
3.8 Gazebo empty world. 28
3.9 Comparison between simulated and real set-up 29
3.10 View of 3D actor model 30
3.11 Feature detection example 31
3.12 Rviz Interface 32
3.13 Mockup of the first version of the rqt-based interface 33
3.14 Instantiation of the first version rqt-based interface 35

3.15 Instantiation of the final version rqt-based interface 36
3.16 CASIR-IMPEP remote lab. 37
3.17 Complete dataflow diagram of the web service. 38
3.18 Face and pose detection in simulated environment 40
3.19 Final simulated environment in action 42
3.20 Final rqt interface with simulator topics. 43

4.1 Overview of the several components of the proposed solution running in operational conditions 47
4.2 New launcher plugin interface. 48

List of tables

1.1 Feature comparison between frameworks. 3
1.2 Availability comparison between frameworks. 3

2.1 Technical comparison between simulators 11
2.2 Support infrastructure of the simulation platforms 11
2.3 Supported robot families 12
2.4 Most used development frameworks available for Unix systems 15

3.1 Guppy Firewire RGB cameras specifications 23
3.2 Microsoft Kinect V1 technical specifications 24
3.3 Motor characteristics. 26
3.4 Gazebo UI *vs* without Gazebo UI benchmark comparison. Percentages and memory usage are relative to the specifications of the main computer 41
3.5 Local vs remote visualisation benchmark comparison, the specifications of both machines are those in section 3.4 44
3.6 Results of the combining the aforementioned tests – data from the main computer. 44
3.7 Results of the simulation and visualisation with the full attention middleware running, data from the main computer. 45
3.8 Internet connection benchmark 46

Chapter 1

Introduction

1.1 Motivation and overall goals

Robot sales have increased to record numbers during the past decade, about 29% to a number of 229,261 units worldwide according to World Robotics 2015 [1], and increasing research in this field has followed. However, robots often are too big to transport, too expensive to replicate, or they may simply not be available to a researcher or developer at a convenient moment in time.

Fortunately, with the increase of computational power, now more than ever, simulation and remote access save time and resources (both physical and budget-related), increasing the productivity of a research team and allowing the community to seamlessly work on the same framework. A robotics simulation consists of a computer generated environment that tries to accurately represent a robotic system by modelling sensor and actuator properties, and also the surroundings it interacts with by modelling real objects and/or actors. There are several advantages in robotic simulation, the most important of which the capability to test new algorithms and routines, reproduce and repeat experiments, generate data under different conditions, neuro-evolve robots and benchmark any of the robot characteristics, without the risk of damaging the real robot [2]. In fact, having the possibility to repeat complex experiments without external variables that may influence their outcome, especially in human-robot interaction (HRI) applications, which depend critically on human subject availability and for which exact repetition is impossible precisely due to this human factor, is a definite advantage[1]. Additionally, there is often a need to open the project to the broader research community, or simply give the development team access from anywhere outside the

[1]This subject will be detailed further in section 2.3.

laboratory. To meet this demand, a recent trend has been the development of remote robotic laboratories [3][2].

On the other hand, the increasing complexity of robotic systems, namely resulting from the number of modules and functionalities it comprises, can overwhelm a developer or user when trying to monitor its operation, and therefore having all of the data organized in a neat and clear fashion is also paramount[3].

The combined set of desired features resulting from this demand and its relationship with potential user types is depicted in Fig. 1.1.

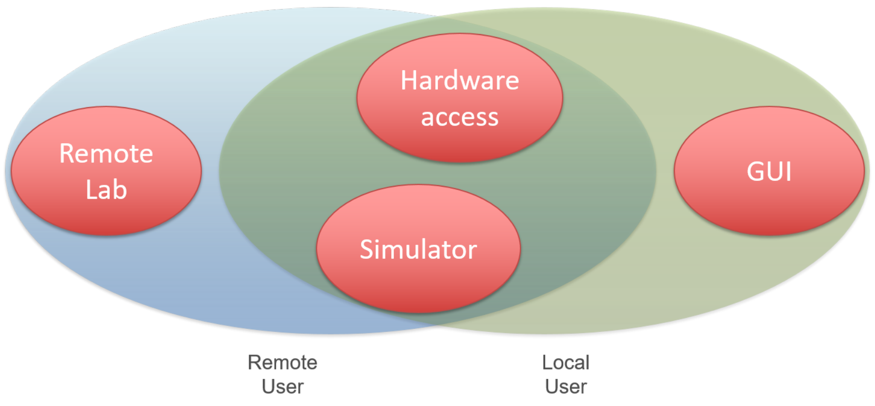


Fig. 1.1 Desired features for most contemporary robotic development frameworks.

The overall objective of the work presented in this dissertation was to endow the robotic system developed during the FCT-funded project CASIR, devoted to studying the effect of artificial multisensory attention in human-robot interaction (more details in section 2.1), with these features, as a follow-up on future work planned in [4]. More specifically, the work presented in this text had the following overall goals: (1) the development of hardware and simulator access to local users, with support of a intuitive local GUI; (2) providing access to these assets to remote users through a remote robotic lab.

1.2 Related work

As the effort of applying a systematic approach to meeting the demand of implementing features such as those presented in Fig. 1.1 is a recent trend, a handful of related works exists – these will be described in the following text.

The Care-O-Bot Research project [5] has a similar architecture to the CASIR framework; however, it deals with a different application scope via a mobile manipulation platform. It

[2]Please find more details on this matter in section 2.5.
[3]Further details will be provided in section 2.4.

includes solutions for local and remote users, including simulation, all of them working through a ROS layer.

The iCub simulator was created to complement the iCub project. It is a very specific simulator with an unique architecture (it uses the same physics engine as Gazebo, however it has its own software), it uses YARP (Yet Another Robot Platform [6]) instead of ROS and a network wrapper for remote access, so it's a custom architecture just for the iCub robot.

Another project, "The Construct Sim" [7], consists of a cloud based tool for remote robotic simulation. This web service contains several simulators, such as gazebo 1.9 to 5.0 using ROS distributions from Hydro to Indigo (In the next chapter ROS will be elucidated to you in detail), Webots and the Darpa Robotics Simulator. In spite having several known robot models, for example Atlas, Reem and Darwin, there is nothing related with artificial intelligence nor an IMPEP model. The Construct Sim has a very limited free user experience, both in simulation time and in computational resources, so in order to properly simulate a scenario one has to resort to the paid services.

A comparative study of these systems in relation to the proposed solution in terms of implemented features and availability is presented in Tables 1.1 and 1.2, respectively.

Table 1.1 Feature comparison between frameworks.

	Remote Lab	**GUI**	**Simulator**	**Hardware**
PR2	yes	yes	yes	yes
Construct Sim	yes	no	yes	no
iCub	no	yes	yes	yes
Care-O-Bot	yes	yes	yes	yes
CASIR-IMPEP	**yes**	**yes**	**yes**	**yes**

Table 1.2 Availability comparison between frameworks.

	Remote Lab	**GUI**	**Simulator**	**Hardware**
PR2	freely available	freely available	freely available	280 000€
Construct Sim	several forms	—	several forms	—
iCub	—	freely available	freely available	250 000€ + tax
Care-O-Bot	on request	on request	on request	on request
CASIR-IMPEP	**free with reservation**	**freely available***	**freely available***	**remote with reservation**

* - actual conditions of licensing (i.e. GPL or other) are to be decided in the future

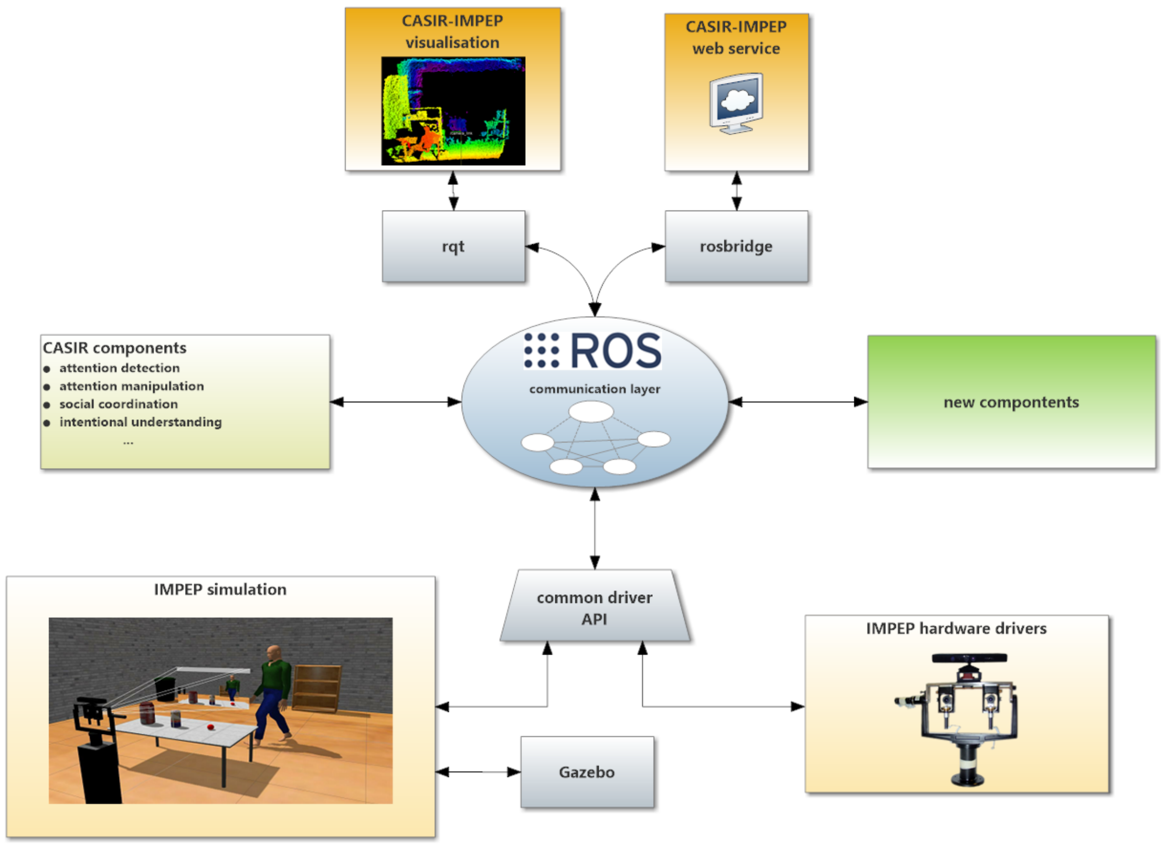


Fig. 1.2 Conceptual diagram for the IMPEP ROS framework for remote 3D visualisation, control and simulation. The modules in orange refer to the contributions of the work presented herewith, namely the simulator represented by the `impep_simulation`, the hardware access that is not only the IMPEP but also it's connection through the `common driver API`, GUI that consists in a rqt-based software and finally the remote lab supported by the CASIR-IMPEP web service .

As can be readily seen, the PR2 and Care-O-Bot have all of the desired features, while the iCub lacks a remote lab and Construct Sim has no GUI nor hardware access.

In terms of availability, while the PR2 and iCub projects have their features freely accessible, hardware can only be accessed via purchase, which in both cases is rather expensive. Construct Sim has several payment options, but no hardware access, for obvious reasons. Care-O-Bot is a different case – the price of every module is subject to a budget analysis by the company.

Conversely, the framework described in this thesis will be developed so as to provide **all the features** of Fig. 1.1 as **freely available**, and, in the case of the remote lab access by a user external to the AP4ISR research team, with reservation of timeslots, all the time **ensuring system and hardware security**.

1.3 Contributions

In summary, the contributions of this work, represented in Fig. 3.1, resulting of the implementation of an integrated framework boasting the features presented in Fig. 1.1, consist of providing the full feature set with the widest availability possible, as shown in Tables 1.1 and 1.2. This will allow the research team to access and develop the attention middleware both locally and remotely, and also make a demonstrator of the CASIR framework available to the wider scientific community.

1.4 Structure of dissertation

This dissertation will be divided into 4 chapters:

- The current chapter provides the motivations, overall goals, related research and expected contributions of this work.
- In **Chapter 2** discusses the background and methods for this work. More specifically, the CASIR project and the IMPEP platform will be introduced, a prior on the Robot Operating System (ROS) will be presented, followed by a study of the state of the art on 3D modelling tools, robotic simulators, UI's and remote experimental laboratories.
- In **Chapter 3**, implementation details and results will be presented. The construction of the ROS framework, composed by the Gazebo based simulator, *rqt* visualizer and *rosbridge* web service, will be described, followed by experimental results providing proof-of-concept.
- Finally, in **Chapter 4** we will draw conclusions and propose future work.

Chapter 2

Background and methods

2.1 An open-source framework for studying artificial perception and attention in HRI

Several studies have shown that humans expect robots to exhibit intentionality and reciprocity in HRI – many researchers believe that both these perceived traits result from unconsciously acknowledging that robots are able to functionally mimic certain human skills, one of which being *attention*, according to psychological studies a tell-tale of intentionality [8].

Following this line of thought, the Institute of Systems and Robotics obtained funding from the Portuguese Foundation for Science and Technology (FCT) for studying the influence of artificial, automatic (involuntary) attentional processes in HRI named CASIR (Coordinated Attention for Social Interaction with Robots)[1]. During this project, a first draft of an artificial attention framework designed to act as middleware between visual and auditory sensor systems and a full-blown cognitive system was developed using ROS[2] – see Fig. 2.1 and [9].

This framework is supported by the IMPEP infrastructure (acronym for Integrated Multimodal Perception Experimental platform) – see Fig. 2.2. For more information about this platform and its hardware, please refer to [4, 10].

[1]FCT Contract PTDC/EEI-AUT/3010/2012, which ran from 15-04-2013 until 31-07-2015.

[2]The ROS development framework will be introduced in the following section.

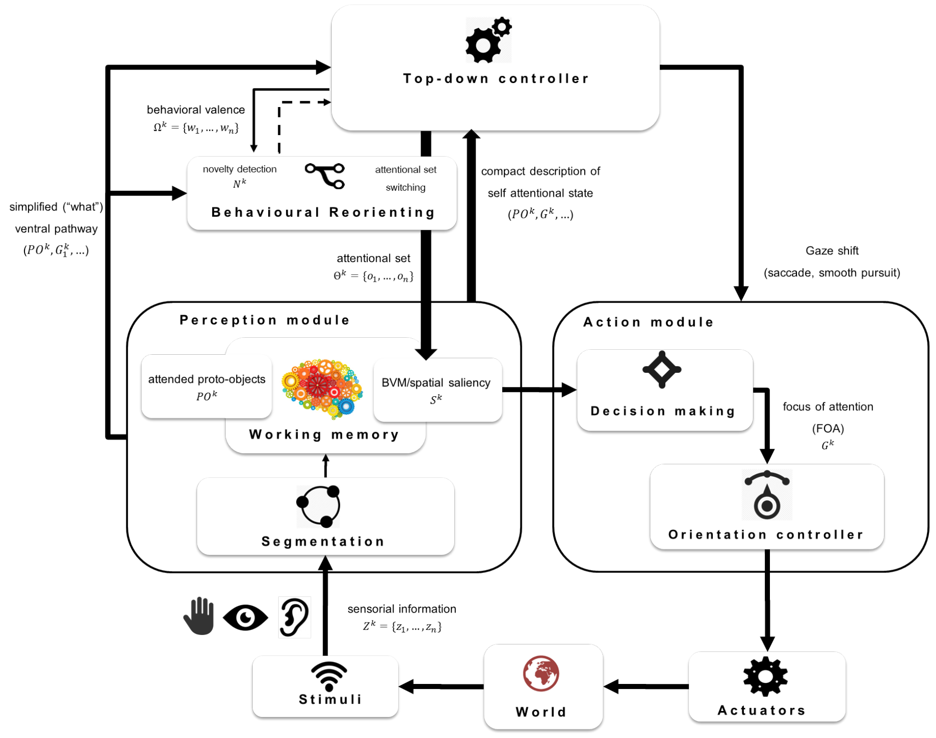


Fig. 2.1 The perception module processes sensory signals to build an egocentric representation of the environment and maintains it in the working memory. The top-down controller generates control signals and sets of relative weights that modulate responses to different features.The action module sends commands to actuators according to the attentional map and to the exploration behaviour informed by the top-down controller. The reorienting module checks for unexpected and behaviourally relevant stimuli, overriding the current attentional set if necessary. (Taken from [9], with permission.)

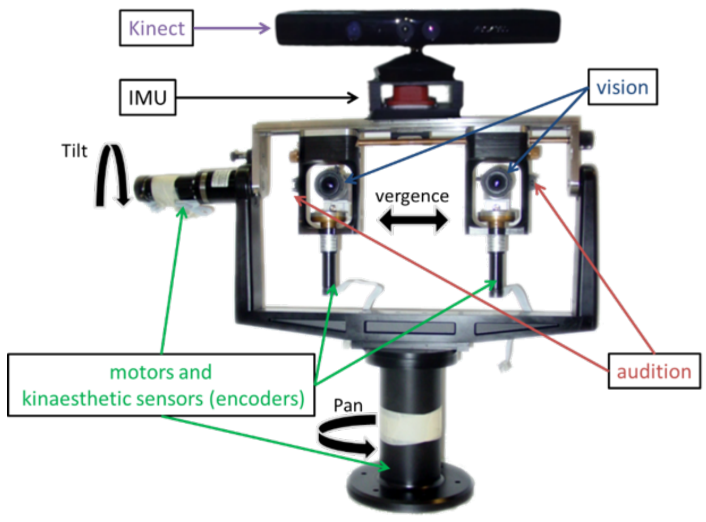


Fig. 2.2 The Integrated Multimodal Perception Experimental Platform [4].

The goals set out for future work on the CASIR-IMPEP ROS infrastructure already involved the overall goals laid out in section 1.1, and consequently this thesis reports on follow-up work of this project within the scope of these goals, as reflected in Fig. 2.3.

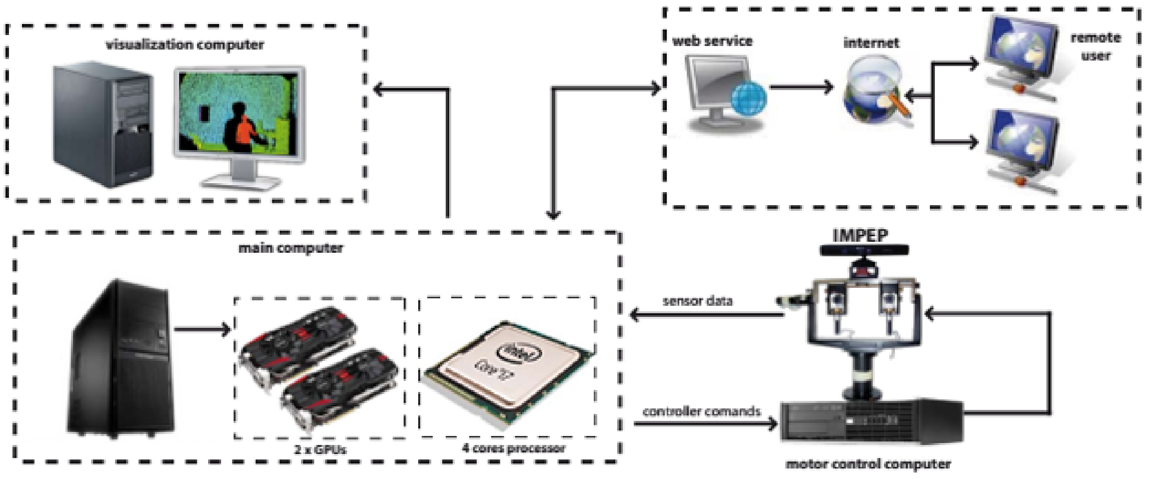


Fig. 2.3 CASIR-IMPEP system architecture overview [4] – only the bottom part of this diagram was originally fully implemented during the duration of the CASIR project, while the top part was planned as a future expansion.

2.2 Building robot applications for the global research community – the Robot Operating System (ROS)

ROS (Robot Operating system[3]) is a flexible framework for writing modular robot software, capable of creating complex and robust behaviour in different types of robotic platforms.

This framework aims to be [11] :

- Peer-to-peer
- Tool-based
- Multi-lingual
- Thin
- Open-Source

[3]In spite of its name, ROS is not an actual operating system in the traditional sense of process management and scheduling.

The ROS framework involves several core concepts, such as *packages*, *nodes*, *topics*, *services* and *messages*, which will be explained next.

Packages are the software organization unit of ROS code. Each *package* can be a collection of libraries, executables and scripts. In a lower level we have *nodes*, in essence executables from a certain *package* that communicate with other *nodes*.

Conventionally a robot control system will comprise many *nodes* – for example one *node* controls the wheels of the robot, while a second *node* is responsible for the navigation and a third for localisation. There are several advantages in this type of modular framework, such as fault tolerance (if a crash occurs in a *node* it will be isolated from the other *nodes*) and reduced code complexity.

The communication between *nodes* is made through *services* or *topics* (basically named buses) over which *nodes* exchange *messages* in a publish-subscribe pattern. To publish/subscribe a *topic* the *node* needs to know both the name and type of the transmitted *message*.

A peer-to-peer system built with ROS will have a ROS Master whose role is to enable individual nodes to locate each other while providing naming and registration services. It also tracks publishers and subscribers to topics.

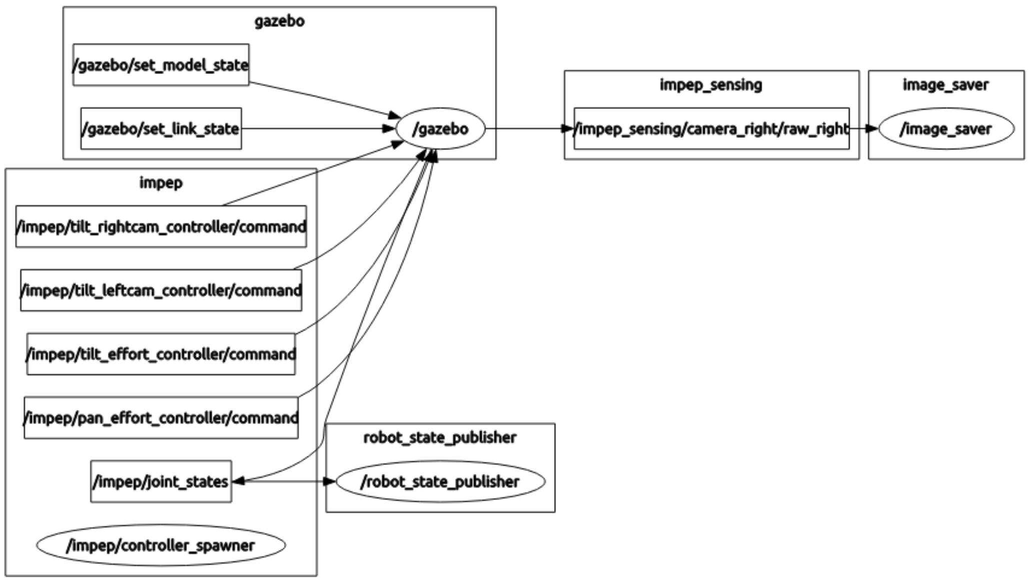

gazebo
/gazebo/set_model_state
/gazebo/set_link_state
/gazebo
impep_sensing
/impep_sensing/camera_right/raw_right
image_saver
/image_saver
impep
/impep/tilt_rightcam_controller/command
/impep/tilt_leftcam_controller/command
/impep/tilt_effort_controller/command
/impep/pan_effort_controller/command
/impep/joint_states
/impep/controller_spawner
robot_state_publisher
/robot_state_publisher

Fig. 2.4 Simulated cameras framegraber node communication.

In the example shown in Figure 2.4, the simulated robot model for IMPEP is publishing from all of its control nodes to the main node of the simulation software Gazebo (explained in the following section). That node is being subscribed by the `impep_sensing` node that receives the camera topic from Gazebo. Finally, the `impep_sensing` node publishes the processed topic to the `image_saver` node.

Dealing with a massive runtime environment may seem daunting; however, with small tools dividing all the tasks, following the modular principle of ROS, it is easier to manage complex systems and increase the stability. Another benefit of the modularity is that it allows ROS to be language-independent – in other words, users can create nodes in C++, Python, Octave and Lisp without losing the possibility of communication between them if the messaging interface is maintained.

ROS has a wide user-base, mostly because it is free and open source – every user can contribute to the framework. As a result, there is an immense number of useful packages available, albeit not always with the clearest documentation; however, documentation tends to be properly updated over time to meet user demands[4].

2.3 Simulation and 3D modelling tools for human-robot interaction

Virtual simulation is one of the most widest accepted recent technologies in robot development. There are numerous software tools used for simulation with big diversity in features (supporting a variety of robotic middleware, available sensors and actuators, and compatible with several types of robots) and also diversity in infrastructure (code quality, technical and community support).

According to [12], currently there are about 40 simulation tools used by the scientific community; however, since this work follows the CASIR project which is supported by ROS, we are only going to compare the main tools compatible with this framework, narrowing the competitors under analysis to Gazebo [13], MORSE [14], V-Rep [15] and Webots [16].

Gazebo was developed by the OSRF (Open-Source Robotics Foundation) and supports several physics engines like ODE, Bullet and DART. Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments, it can also be easily extended with new features due to having a modular and plugin-based structure [13].

Morse is a simulator for academic robots with the primary objective of simulating realistic 3D environments, both indoor and outdoor with a single robot or a swarm of tenths of robots. Its rendering is based on the Blender Game Engine and uses the physics engine Bullet [14].

V-Rep, by Coppelia Robotics, is a software platform for robot simulations with a distributed control architecture in which each object can be controlled individually. It supports several physics engines like ODE, Bullet and Vortex [15].

[4]Some of these packages are going to be mentioned later in this dissertation, such as *rviz* and *rqt*.

Webots by Cyberbotics is a development environment for simulating mobile robots, in which the user can design his/her own complex robotic setup, with one or several robots. Webots uses a custom version of the ODE as its physics engine [16].

A comparison of their technical characteristics is presented in Table 2.1, while a comparison of the support infrastructure provided by each developer and/or community is presented in Table 2.2. Moreover, only Gazebo provides the percentage of coverage from function and branch testing (52.9% and 44.5% respectively) as seen in the Gazebo website [13].

Table 2.1 Technical comparison between simulators

Software	Programming Language	Supported Formats	Extensibility	Robotic Middleware	User Interface	Headless Simulation
Gazebo	C++	SDF/URDF	Plugins (C++)	ROS, Player, Sockets	GUI	Yes
MORSE	Python	unknown	Python	Sockets, YARP, ROS, Pocolibs, MOOS	Command Line	Yes
V-Rep	LUA	OBJ, STL, DXF, 3DS, COLLADA, URDF	API, Addons, Plugins	Sockets, ROS	GUI	Yes
Webots	C++	WBT, VRML'97	Plugins (C++), API	ROS, URBI, NaoQI	GUI	Yes

Table 2.2 Support infrastructure of the simulation platforms

Software	Mailing List	API Documentation	Public Forum	User Manual	Issue Tracker
Gazebo	Yes	Yes	Yes	Yes	Yes
MORSE	Yes	N/A	No	Yes	Yes
V-Rep	No	Yes	Yes	Yes	Unknown
Webots	No	Yes	Yes	Yes	Yes

In terms of available actuators and sensors, all of them have the same support: both generic kinematic chains and force-controlled motion actuators. Regarding sensor types, all of these simulation environments support modelling odometry, IMU, collision, GPS, cameras (monocular, stereo and depth), laser scanners (2D and 3D), and boast a full Microsoft Kinect model.

The robot families supported by each simulator are reported in Table 2.3.

Table 2.3 Supported robot families

Software	Ground mobile robots	Aerial robots	Underwater robots	Robotic arms	Robotic hands	Humanoid robots	Human avatars
Gazebo	Yes	Yes	Yes	Yes	Yes	Yes	Yes
MORSE	Yes	Yes	Partial	Partial	No	No	Yes
V-Rep	Yes	Yes	No	Yes	Yes	Yes	Yes
Webots	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Analysing this objective information we can conclude that Gazebo and Webots stand out from the 4; however Gazebo is slightly ahead in terms of support infrastructure.

Finally, according with a survey about simulation frameworks based on user feedback[12], Gazebo is the most used and known simulation software amongst the population in study, mostly for the community around it, being open source and for being the most suitable for research in a subjective point of view.

In order to simulate an environment, one needs to build the models using 3D modelling tools, and in this case (robotic applications) these tools must be able to work with the COLLADA format (easily parsed to SDF or URDF), as it is a type of format that contains all of the 3D information and the materials, being the only compatible with Gazebo to do so [17]. The most relevant tools in this case are Maya, 3DStudioMax and Blender. This 3 solutions are very similar in features, so the criteria for comparison should be price, ease of use and operative system compatibility.

Regarding the first criterion, the open source software Blender is superior comparing with its competition that are subject to licensing costs of thousands of euros. As for ease of use, Blender is also superior, since 3DStudioMax and Maya are professional software tools used in the game develop and animation industries – due to the simplicity of the modelling demands of the work reported in this thesis, and without the use of complex animations, Blender is the most suitable solution, with no added difficulty for the user. Finally, both Maya and Blender are compatible with Linux, OSX and Windows, while 3DStudioMax is Windows only.

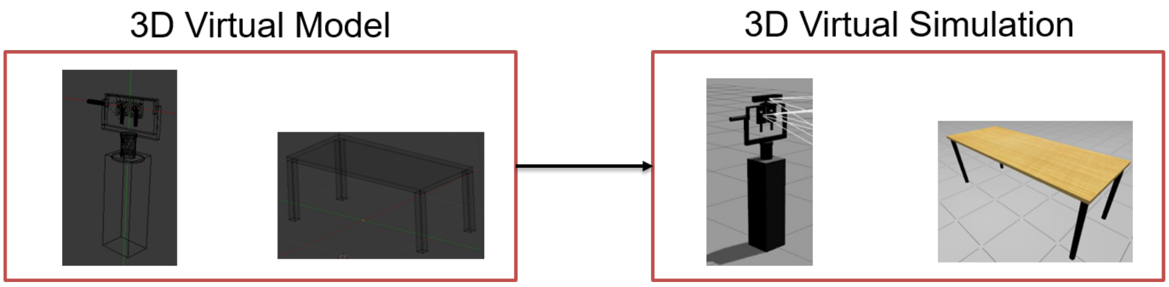


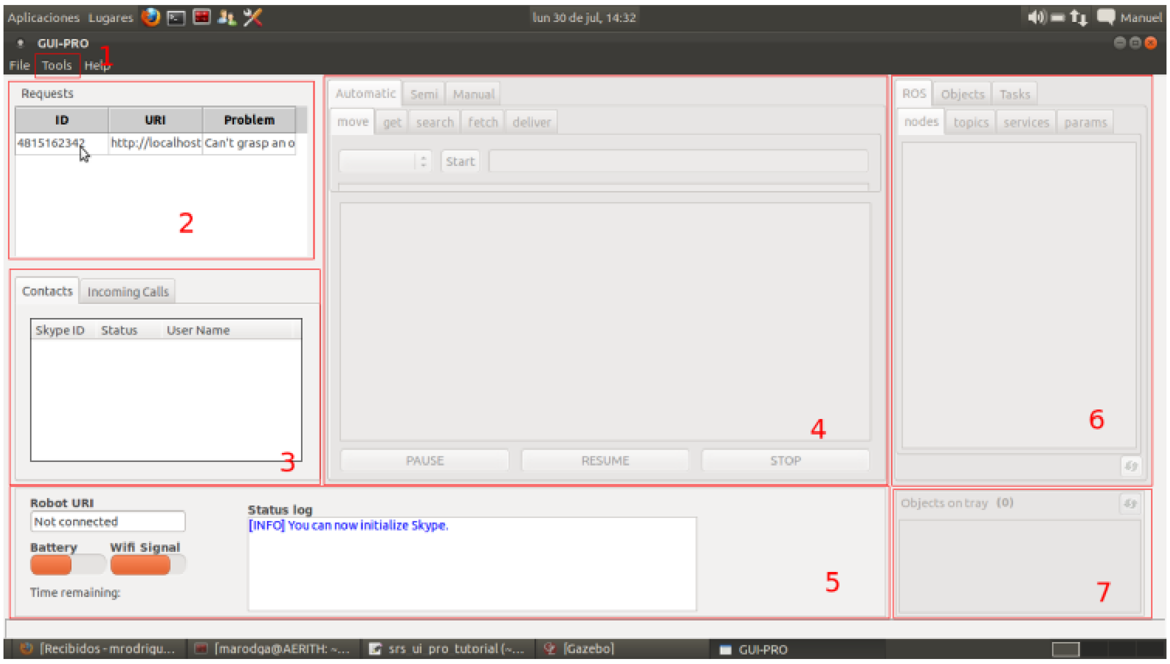
Fig. 2.5 IMPEP and table model created in Blender and inserted into a Gazebo world.

Considering all this information and that previous existing models of IMPEP were already made in Blender, this software was chosen to build the final and current version of the IMPEP model used in the simulator. The development pipeline for modelling in this work can be seen in figure 2.5.

2.4 Graphic user interfaces for robotic applications

As described in section 1.1, with the growth in complexity of robotic systems, a user or even a developer can easily become overwhelmed with information, thus becoming more difficult to have an overall view of the operations, control the system and acquire and handle data. In order to organise all of this information and give the desired control to the use, the graphic user interface must be designed in order to be simple and intuitive. We are surrounded by human-machine interfaces (HMI) in our daily life, ranging from a LED showing the on/off status of a TV to the interface of an automatic register in a supermarket.

Applying HMI to robotics is as important as the system itself – it is critical that the user possesses and and is familiar with the right tools to work with the system.



ID	URI	Problem
4815162342	http://localhost	Can't grasp an o

```
[INFO] You can now initialize Skype.
```

Fig. 2.6 Care-O-Bot3 dashboard (srs_ui_pro) created in wxWidgets. In 1 we have the Tools menu, in 2 a request events panel, 3 the skype panel for communications, 4 actions panel, 5 status panel, 6 options panel and finally 7 with objects on tray panel. — publicly available online, taken from [18] .

As with any software development project, there are several choices to make, from the programming language to the development framework itself. The most relevant frameworks are listed in Table 2.4. Among these, Qt [19] or wxWidgets [20] (an example can be seen in Fig. 2.6) allow the developer to create a complex and attractive GUI completely personalised to his/her system.

However, in recent ROS distributions there is a tool named *rqt* that is basically a framework for plugin development. In rqt, a developer can build his/her own perspective from plugins of all the existing GUI tools in ROS, namely *image viewer*, *terminal*, *2D plot*, *node and package graphs*, *pose viewer* and even *Rviz* itself [21]. If the available plugins are not suitable for the needs of a project, the developer can either edit an existing plugin or even create his/her own plugin (either in C++ or Python). All of this information is conveniently summarised in Fig. 2.7.

Table 2.4 Most used development frameworks available for Unix systems

Toolkit name	Operative System compatibility	Programming language	License
Qt	Windows, OSX, Unix	C++	LGPL
wxWidgets	Windows, OSX, Unix	C++	WxWindows license
Ultimate++	Windows, Unix	C++	BSD
GTK+	Windows, OSX, Unix	C	LGPL

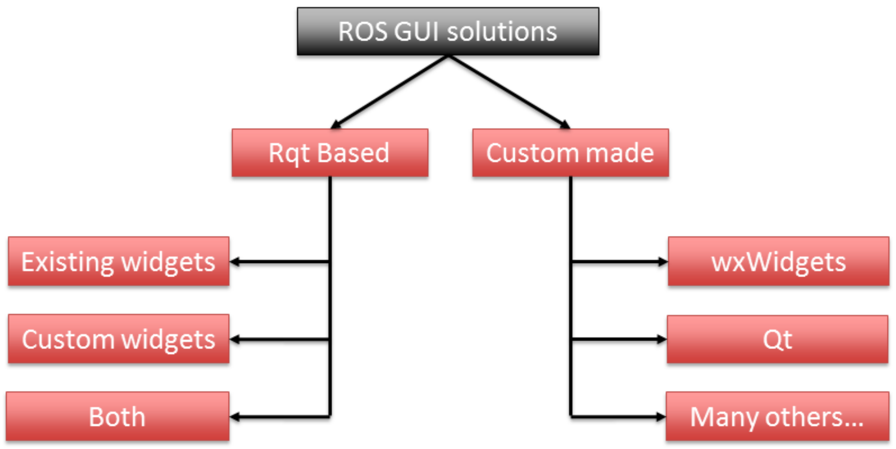


Fig. 2.7 HMI development frameworks supporting ROS.

2.5 Remote robotic experimental laboratories

Nowadays with the emergence of robot middleware and systems, robotics have drastically advanced. However there are some restrictions that can hinder this evolution, namely the price of the equipment, limited experimentation time and even availability of resources. In order to deal with these restrictions, a recent trend has emerged – remote experimental labs. These allow remotely sharing robot middleware infrastructures in a modular way with the broader scientific community, making it easier to compare and contribute to the research of others.

Many robotic researchers have resorted to web technologies for remote robot experimentation, data collection and HRI studies. There are examples of remote access and control of robots from as early as 1995, in the case of [22].

Unlike ROS's architecture of distributed publishers and subscribers, remote experimental labs use a client-server architecture. This kind of systems divide the workload, with dedicated servers being responsible for providing services requested by each client. The arrival of new web technologies such as HTML5 and JavaScript makes it possible for developers to create appealing and sophisticated interfaces. With the use of protocols such as *Rosbridge*, the communication between a web browser and ROS can be made through data messages contained in JSON. JSON is a lightweight data-interchange format with which Rosbridge communicates with ROS itself [23].

In the code snip presented in Listing 2.1, we can witness this concept in use via a Javascript example of a topic publication defining the name and message type as well as the message itself, composed of linear and angular velocity vectors.

```
//----------------------- Publishing a Topic -----------------------//

  var cmdVel = new ROSLIB.Topic({
    ros : ros,
    name : '/cmd_vel',
    messageType : 'geometry_msgs/Twist'
  });

  var twist = new ROSLIB.Message({
    linear : { x : 0.1, y : 0.2, z : 0.3},
    angular : { x : -0.1, y : -0.2, z : -0.3}
  });

  cmdVel.publish(twist);
```

Listing 2.1 Rosbridge Topic Declaration and message publishing

With the above style of coding, the developer can easily create widgets to incorporate functionalities such as teleoperation controllers, topic and node listings in similar to the PR2 remote laboratory seen in Fig. 2.8.

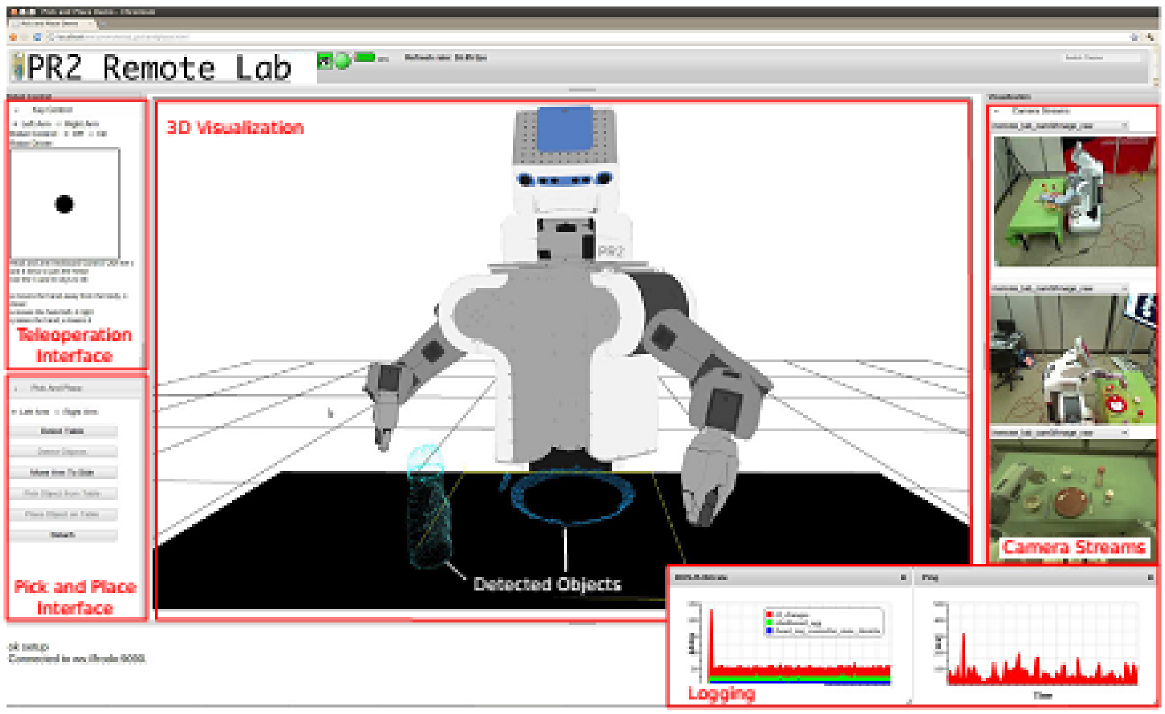


Fig. 2.8 PR2 remote laboratory using *Rosbridge* and *mjpeg_server* [24]. In this remote lab we can se a teleoperation interface, bandwidth logs and camera streams. Figure from [25].

In order to address the development of a great variety of applications, RobotWebTools [26] created three libraries to help with web-based HRI, namely *roslibjs*, *ros2djs* and *ros3djs* for visualising either 2D or 3D ROS data types, allowing the user to see both RGB messages and point clouds in a HTML `<canvas>` element.

Besides seeing ROS information in the form of images, we also need to transmit them over *rosbridge* – for this end, the ROS package named *web_video_server* is used. Within this package there are two streaming options for the developers to use. The first option is based on the deprecated package *mjpeg_server*, and consists in converting the video stream from the desired ROS topic into a mjpeg stream (a sequence of jpeg images), this stream can then be embedded into any HTML `<img>` tag. The second option consists in coding the video with the VP8 codec, as stated in [27] – this codec has a low computational complexity and high compression efficiency. To embed the stream using this codec, the developer has to use the new `<video>` tag in HTML5, this creates a limitation in the web browsers supported (typically Chrome and Firefox have a good support). This option also has an unavoidable delay in the transmission due to the buffering within the video codec itself.

Chapter 3

Implementation and results

3.1 Putting it all together – a ROS framework for the CASIR-IMPEP platform

The expected outcome of this work is a unified ROS-supported framework designed so as to attain the objectives laid down in section 1.2, allowing the CASIR attention middleware represented in Fig. 2.1 to be used within the context defined by those objectives and the use of the IMPEP platform. Additionally, it is a desired property that this framework be easily adaptable to conform with any robotic head with some or all of the same characteristics as IMPEP, so as to be used with any robotic platform with innate multisensory attention capabilities. A diagram summarising these desiderata is shown in Fig 3.1.

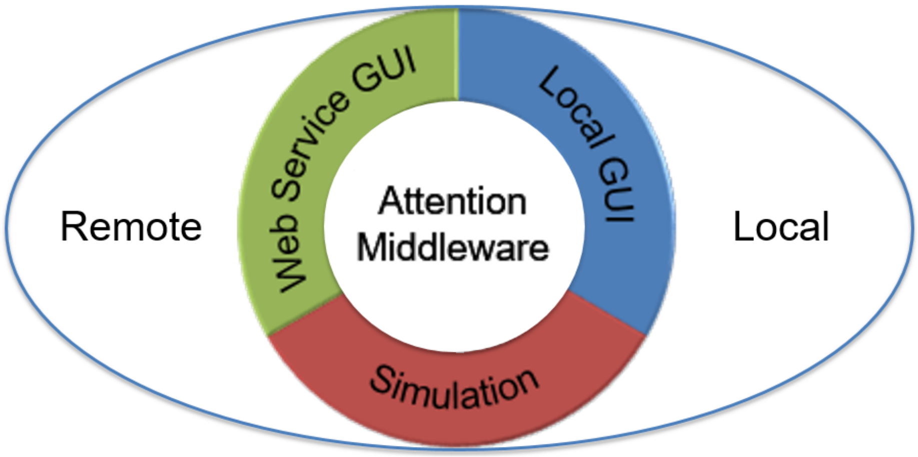


Fig. 3.1 Framework goals for the IMPEP platform, addressing both local and remote user needs while providing a viable bridge to the existing middleware.

This system, detailed in the diagram of Fig. 3.2, we can have either the simulated or the real version running at once, both of them publishing sensor information to the same ROS topics (a concept represented by the Common Driver API module in the diagram). The published topics can be subscribed by the attention middleware nodes or seen directly by the remote and local users through the respective GUIs.

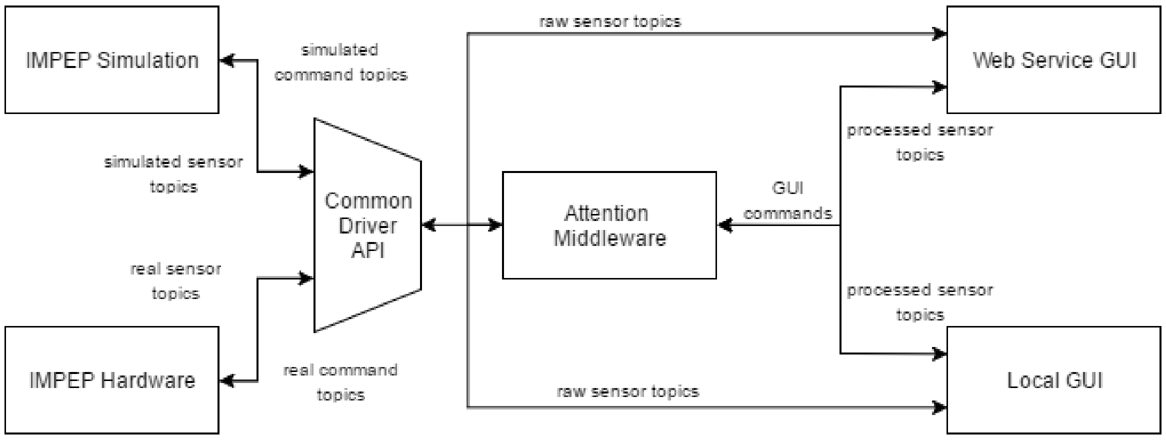


Fig. 3.2 Simplified system dataflow.

Commands, on the other hand, follow almost the inverse path, the only difference being the non-existence of a direct connection between the GUIs themselves and the physical, as well as virtual, actuators. Manual control of both versions of the robot can be made through a node in the attention middleware using terminal commands, which can be sent within the local GUI (see section 3.4).

Since the CASIR was built on an older ROS distribution – ROS Fuerte, the last distribution to use rosbuild (a CMake based build system for ROS) as the build method – a full scale migration had to be done to the entire middleware. This was due to an incompatibility with recent tools selected for this project (e.g. gazebo version, rqt itself), the previous system didn't have the minimum capabilities to execute such framework in the desired fashion.

With this migration, the CASIR-IMPEP middleware was now up-to-date and running with ROS Indigo, one of the most used ROS distributions, now using catkin. This successor of rosbuild combines CMake macros and Python scripts to provide improved functionality over CMake's normal work flow. Since it was a prerequisite for the migration, the operative system itself was upgraded from Ubuntu 12 to 14.04, and the CUDA[1] and OpenCV APIs,

[1]A GPU programming environment by NVIDIA. The CASIR-IMPEP middleware uses this API indirectly through OpenCL, an open-source GPU programming environment. Please refer to [28] for more information.

and also all hardware drivers, were updated accordingly. In a nutshell, this migration allowed us to use more recent and powerful tools and ROS packages to create the desired modules.

Throughout the remainder of this chapter each of the developed modules are going to be described in detail, both in terms of implementation and behaviour.

3.2 Implementation details for the Gazebo-based simulation package

As seen in section 2.3, Gazebo stands out from the abundant variety of robotic simulator options that exist in the research community nowadays. However there are also multiple versions of Gazebo that differ from one another in features, support and system compatibility.

In the previous version of the system (as mentioned previously, using ROS Fuerte and Ubuntu 12), there already existed a tentative ROS package named *simulator_gazebo*, a preliminary, crude attempt at a Gazebo project. After the migration, we were able to use a more recent and standalone version of Gazebo. The official Gazebo version for ROS Indigo is 2.2 from 2013-11-07; however version 4.0 was the last released under the ROS Indigo distribution and added very important features, not only for the work at hand, but also for future work. More specifically, Gazebo 4.0 is the first to include a *human model with animation capabilities*, an essential tool for generating HRI simulations, and features such as copy-and-paste models via GUI. It also includes clear documentation, and, most importantly, Razer Hydra and Oculus Rift support, which are very interesting for future work on simulation.

In our case, there were three main packages to create in order to build a complete robot model that is fully compatible with ROS: `impep_gazebo`, `impep_controller` and `impep_description`.

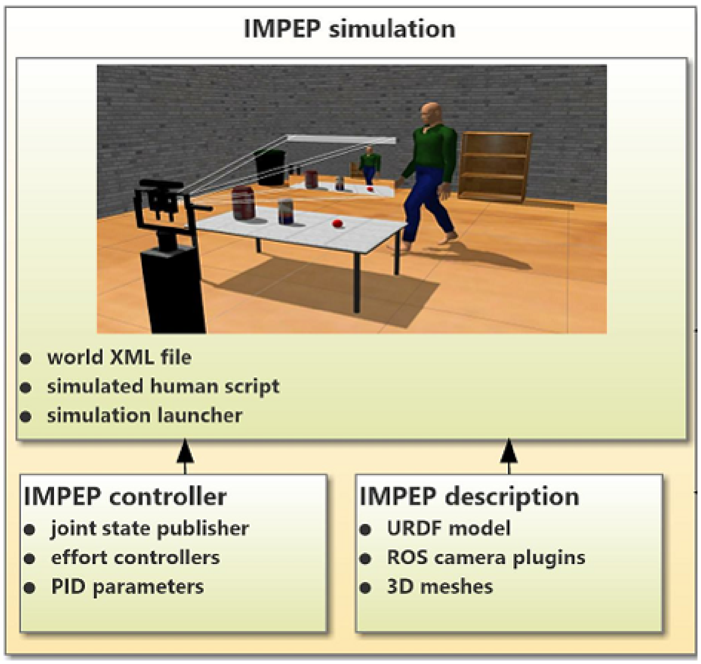


Fig. 3.3 IMPEP model packages for simulation.

As seen in Fig. 3.3 the main package is `impep_gazebo`, which includes the world file (addressed in more detail in subsection 3.2.3), the avatar scripts (section 3.2.4) and the ROS launch file.

The `impep_description`, package is responsible for the robot model itself and contains the 3D meshes of each individual part (modelled using Blender – see section 2.4) which will be the links of our robot. The links are static parts, which connect the joints of an robot together, while the joints are parts of the robot which actually bend or move, together these components form a kinematic chain. Using the meshes we can build the URDF (Unified Robot Description Format) model, which is an XML format describing the links and joints of the robot, defining the geometry, position and collision mesh of each 3D component, and consequently resulting in models such as represented in Fig. 3.4.

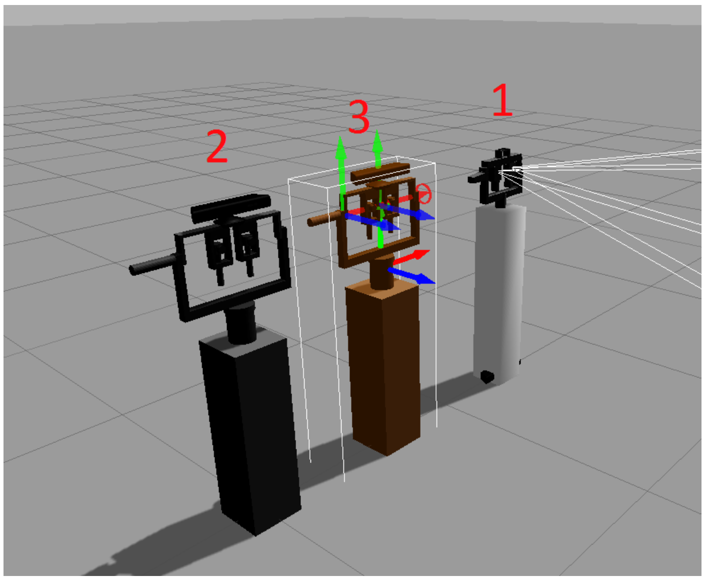


Fig. 3.4 IMPEP virtual model evolution. Model (1) was the pre-existing, preliminary IMPEP model: it had no controller, no defined collisions, and did not faithfully represent its geometry. Model (2) is the upgraded physical model of IMPEP, completely to scale in terms of mass and dimensions. Finally, model (3) shows the addition of the collision mesh and joint referentials, which all together make up the final version of the simulated IMPEP.

Natively in ROS Indigo there is a tool named *check_urdf* that, if the URDF file is well built, parses the file and prints a description of the resulting kinematic chain[2].

With the URDF file successfully created and parsed, the next step was to address the sensors and actuators of the model. The models for the former, described in the following section, are also included in the `impep_description` package. For the latter, a third and final package, `impep_controller` was developed. This package is responsible for joint controllers and will be addressed in section 3.2.2.

3.2.1 IMPEP simulation – sensors

The IMPEP model has three visual sensors: two RGB cameras, and a Microsoft Kinect sensor: These were modelled as faithfully as possible in the URDF IMPEP model.

More specifically, the RGB stereovision set-up, mounted so as to allow pan, tilt and version using IMPEPs actuators, consists of a pair of Guppy F-036 [29]. These are ultra-compact VGA cameras for high contrast applications that support HDR (High Dynamic Range) mode. Further information and specifications are presented in Table 3.1.

[2]The output of this tool for the upgraded IMPEP model is included in Appendix A.

Table 3.1 Guppy Firewire RGB cameras specifications

Guppy F-036 Specifications	
Interface	IEEE 1394a - 400mb/s
Resolution	752 (H) x 480 (V)
Sensor	OnSemi MT9V022
Sensor type	CMOS Progressive
Max frame rate at full resolution	64 fps
ADC	10 bit
Bit depth	8 bit
Mono modes	Mono8
Raw modes	Raw8
Power requirements	8V to 36V
Power consumption (@12V)	<2W
Mass	50 g
Body dimensions (L x W x H in mm)	48.2 x 30 x 30 (including connectors)

Two types of important characteristics in terms of simulation modelling are presented in this table, the first of which being physical characteristics of the camera, such as mass and body dimensions. In order to create a faithful virtual representation of the cameras, their 3D model followed these parameters exactly. While the mass has to be inserted in the URDF and can be altered with the change of a variable, the dimensions on the other hand must be changed in the 3D mesh itself, implying a change of the corresponding Blender model.

The second type of relevant characteristics are the technical specifications of the camera, namely its frame rate, resolution and bit depth. In order to create a virtual camera with these specifications, a Gazebo sensor with the type "camera" was added and a Gazebo-ROS plugin named *libgazebo_ros_camera.so* attached to both right and left camera lens models. Next, image height and width as well as the frame rate were defined. The above mentioned plugin is responsible for the publication of the camera data to a *rostopic* specified in its parameter definition.

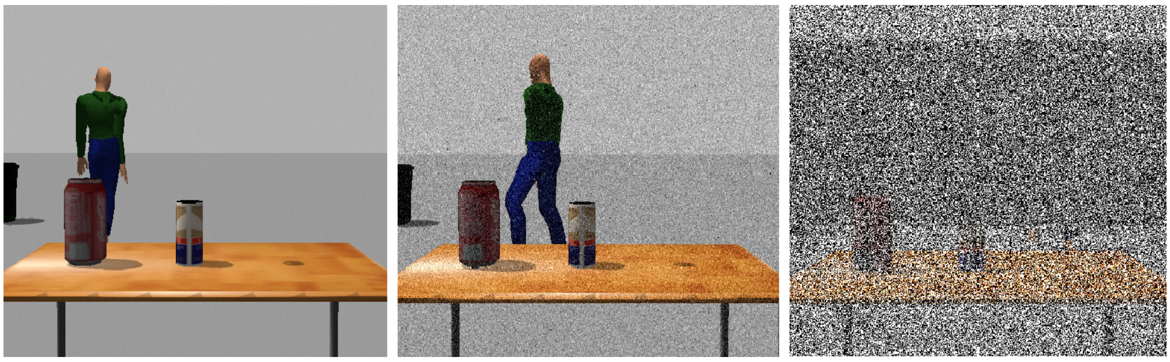


Fig. 3.5 Demonstration of the effect of Gaussian noise modelling on simulated IMPEP camera image quality. Gaussian noise with zero-mean was applied in all cases: from left to right, the standard deviations are 0.007, 0.1 and 1, respectively.

Finally, the effect of Gaussian noise was also modelled in order to simulate residual imperfections intrinsic to every real camera. An example of the effect of different standard deviations can be seen in Fig. 3.5 – as can be readily seen, this is an important variable that can heavily influence the outcomes of image processing.

Having successfully integrated the virtual camera sensors into the IMPEP model, the next step was to add the depth camera, the Microsoft Kinect V1 RGB-D sensor. There already exists a Microsoft Kinect 3D model natively available in Gazebo that follows the body dimensions presented in Table 3.2; however, the remainder of the parameters had to be inserted into the model by hand.

Table 3.2 Microsoft Kinect V1 technical specifications

Microsoft Kinect Specifications	
Depth image size	VGA 640 (H) x 480 (H)
Operation Range	0.6 meters ∼ 4.6 meteres (at least)
Max frame rate at full resolution	30 fps
Power requirements	12V
Power consumption idle (@12V)	∼ 3.3W
Power consumption active (@12V)	∼ 4.7W
Body dimensions (L x W x H in mm)	63.5 x 279.4 x 100 (including connectors)
Mass	1370 g
VGA Horizontal FOV	62°
IR Horizontal FOV	58°

As with the RGB cameras, for the Kinect we introduced a virtual sensor in the model, referenced to the Kinect link, this time with the type "depth", in which we defined image resolution, frame rate, FOV (field of view) and its operating range (minimum and maximum). For the simulated depth camera to communicate with ROS, the `libgazebo_ros_openni_kinect.so` plugin was used, allowing us to define the camera namespace and topics, with the plugin fully configured and tested we had the output seen in figure 3.6.

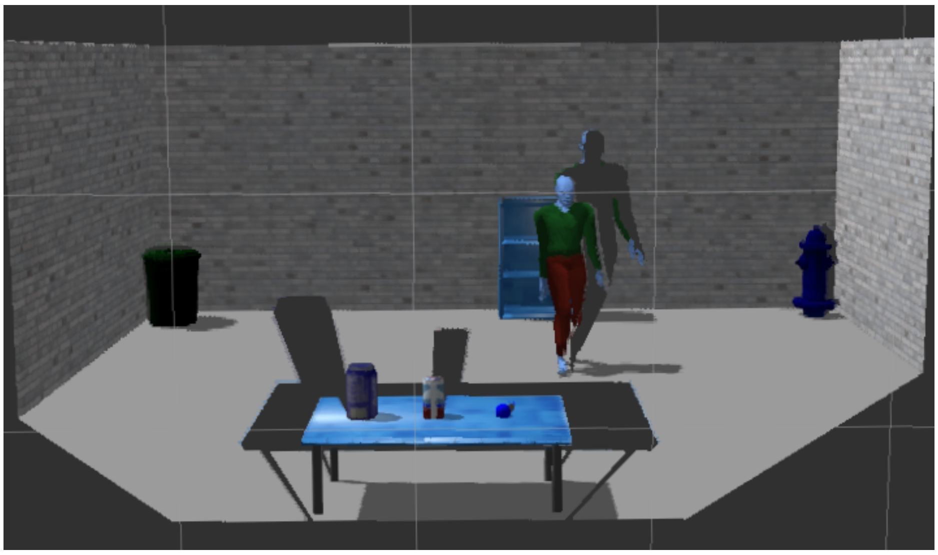


Fig. 3.6 Point cloud generated by the virtual Kinect sensor, transmitted to ROS through a topic. Visualization in *Rviz*.

After implementing all the visual sensors in the model, the kinaesthetic sensors, i.e. the limit/end of movement sensors, were modelled. In the real IMPEP, this limit signal is sent by several micro-switches, both for the pan and tilt movements of the head itself, as well as for the pan movement of the individual cameras.

In order to implement a virtual version of this feature, we were forced to restrict the range of motion in certain joints; as this relates also to the virtual actuators we will explain the specifics of this implementation in the next section.

3.2.2 IMPEP simulation – actuators

IMPEP was built with two different types of DC motors – two PMA-11A-100-01-E500ML motors (one for pan one for tilt) and two PMA-5A-80-01-E512ML motors (one for each camera axis) all from Harmonic Drive (further information about the motors in table 3.3 and [30]).

Table 3.3 Motor characteristics.

	Unit	**PMA-5A**	**PMA-11A**
Maximum output torque	T_{max} [Nm]	0.39	5.0
Maximum output speed	n_{max} [rpm]	180	100
Maximum motor speed	n_{max} [rpm]	9000	5000
Rated motor speed	n_N [rpm]	4500	3500
Armature Resistance	R [Ω]	7.4	1.58
Armature Inductance	L [mH]	0.3	0.29
Weight without brake	m [kg]	0.1	0.5
Moment of inertia without brake	Jout[$x10^{-4}$ kgm^2]	3.68	109
Dimensions	[mm]	100 x 100 x 3	150 x 150 x 6
PID	units	[100, 0.1, 10]*	[100, 0.1, 10]*

* - generic PID, this works with extra intrinsics joint parameters inserted in each motor joint of the XML file such as damping

To emulate virtual motors using URDF, all joints corresponding to each motor must be appropriately defined. For example, a joint that only connects two links with no movement (in our example, the stand of IMPEP and the base of the head) will be stated as“fixed”, while a joint connecting two links that will rotate in one of the referential axis (e.g. the base of the head and bottom of the frame where the pan movement occurs) will be stated as a “revolute” joint.

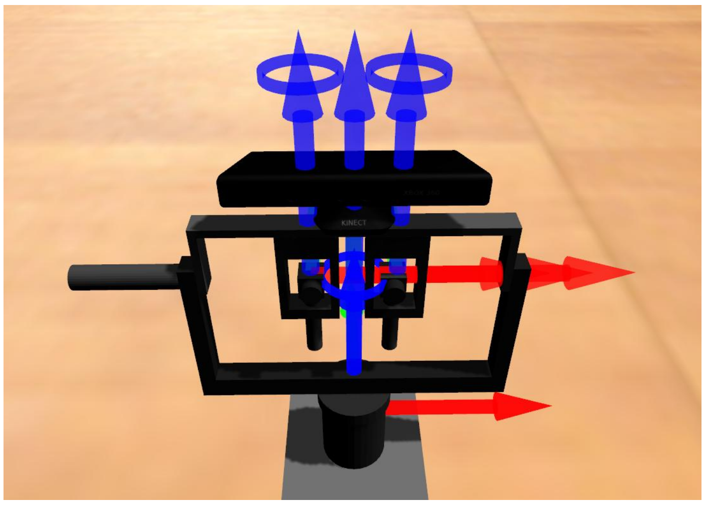


Fig. 3.7 Revolution joints emulating pan and tilt motors. Circles represent around which axis the rotation occurs.

The differentiation between fixed and revolute joints will result from the low-level foundation implementing the virtual actuators according to the technical specifications of each motor.

As mentioned at the end of section 3.2.1, the implementation of end of movement sensors consists in creating an upper and lower movement limit in the revolute joints. For example, the head pan joint will be limited to movements between -1.57 to 1.57 Radians (or approximately -90º to 90º). With these restrictions in place, the virtual IMPEP will have the same range of motion as the real one in every moving joint.

In addition to movement, effort and velocity limits were also implemented, not only to emulate the safety mechanisms of the real IMPEP, but also to further approximate the behaviour between both versions of the robot.

With all the limits and joint parameters defined, the `impep_controller` ROS package was developed using the `libgazebo_ros_control.so` plugin in order to allow communication between Gazebo and ROS, similarly to the camera plugins. This package is responsible for several actions:

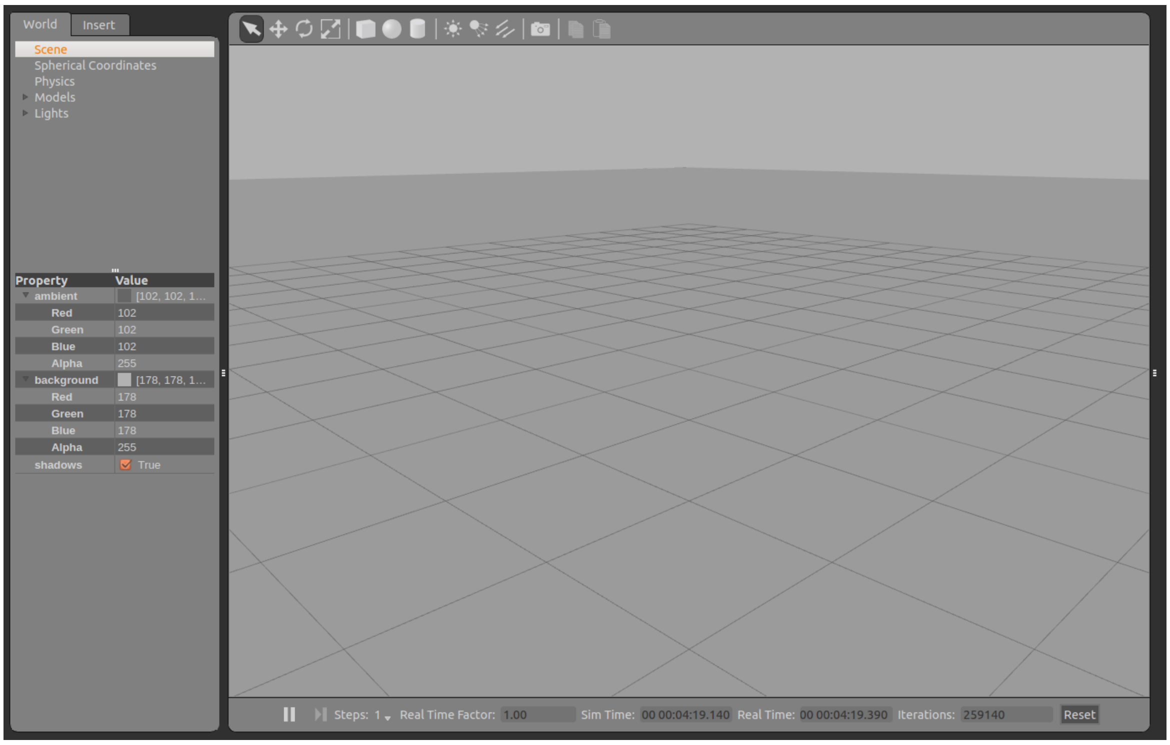
- 1^{st} - Corresponding each joint to an actuator interface, while defining the mechanical reduction and type of Joint Command Interface
- 2^{nd} - Defining a specified rate in which the joint states will be published
- 3^{rd} - Corresponding each PID coefficients to the respective effort controller
- 4^{th} - Converting joint states to TF transforms for *rviz* and other ROS tools

3.2.3 Environmental simulation

In a robotic simulation, modelling the virtual surroundings that constitute the experimental environment the robotic platform will operate on is as crucial as modelling the platform itself. This requirement was taken particularly seriously, given that the CASIR-IMPEP framework is meant to be used in the highly unstructured and dynamic context of HRI.

Rigorously applying simulation terminology, this environment in a Gazebo simulator is called “world”, and this term is used to describe a collection of not only robots and objects, but also global parameters including the sky, ambient light, and physics properties.

The foundation to build a custom world is the Gazebo preset named *empty.world*, shown in Fig. 3.8, which is used by the developer to populate with objects and change textures according to the desired specifications.



Property	Value
ambient	[102, 102, 1...
Red	102
Green	102
Blue	102
Alpha	255
background	[178, 178, 1...
Red	178
Green	178
Blue	178
Alpha	255
shadows	✓ True

Steps: 1 Real Time Factor: 1.00 Sim Time: 00 00:04:19.140 Real Time: 00 00:04:19.390 Iterations: 259140

Fig. 3.8 The default Gazebo world, called empty world, in which a set of default parameters, such as gravity and ambient lighting, are already predefined.

When defining the physics parameters of the simulator, a great care was invested in configuring parameters such as gravity and update rate in order to achieve the most realistic result possible while keeping real-time performance.

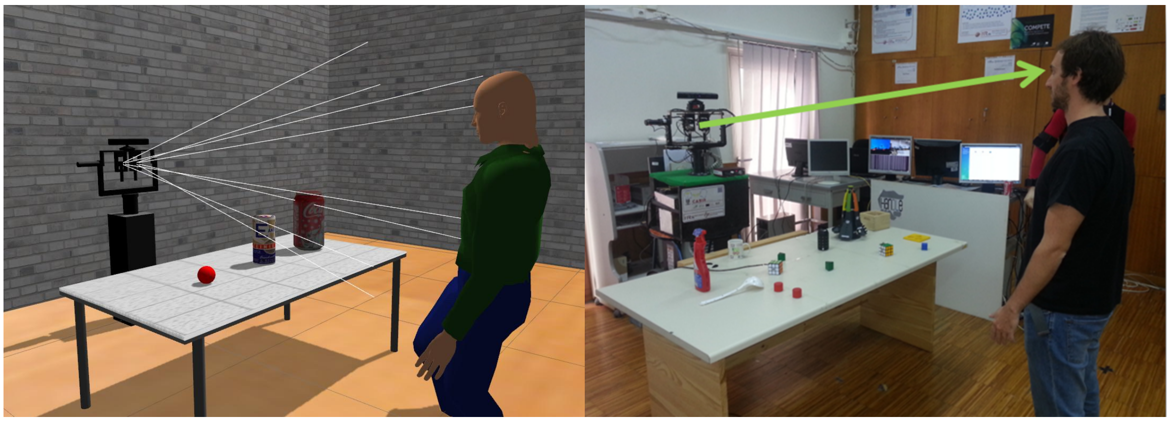
In this respect, achieving

$$realtimefactor = 1 \tag{3.1}$$

means

$$\Delta T_{simulated} = \Delta T_{real} \tag{3.2}$$

In the case of our project, it was important to model a virtual experimental environment that would resemble the environment of our real-life lab, specifically the work area for HRI, where the user has a table with several objects of different colours designed to purposefully study attention mechanisms.



(a) Simulated set up (b) Real set up

Fig. 3.9 Comparison between simulated and real set-up for HRI.

In Fig. 3.9 we have a direct comparison between the work area of the simulated and real IMPEP. Some key variables like distance to the table, table-top and experimental object colour were approximated as much as possible in the simulated environment. The rest of simulated laboratory was populated with roughly the same kind of static objects (e.g. tables and bookshelves); some additional objects in the room were purposely modelled as being red, so as to add perceptually salient entities, which can be used as potential distractors in attention studies [31].

Objects were introduced into our custom world through the Gazebo GUI with the desired position and orientation, using the UI features present in Gazebo 4.0. However the change of textures, physics and fine tune of parameters were made directly into the world XML file, `room_only.world`, which will be called in the main launch file of the package `impep_gazebo`.

3.2.4 Avatar and interaction simulation

Being able to repeat complex experiments in real and simulated human-robot interaction (HRI) settings is one of the objectives of this work. Therefore, a very important feature in this simulator, described in this section, is the availability of a virtual human model with movement and interaction capabilities. Fortunately, as stated at the beginning of section 3.2, the chosen version of Gazebo was the first to include such a feature.

In Gazebo, animated models are called *actors* – as the name indicates, these extend common models by adding animation capabilities.

These animations may be divided into three classes:

1. Skeleton animations in which the motion is relative between links in one model. Gazebo supports two different skeleton animation file formats, COLLADA (`.dae`) and Biovision Hierarchy (`.bvh`).
2. Trajectory animations in which the body moves as a whole through a series of predefined waypoints.
3. A combination of both that allows for a skeleton animation to move around the world.

Animations are relevant if a project requires entities following predefined paths in simulation without being affected by the physics engine; in other words, if the actor's body is not supposed to have a collision mesh or be influenced by gravity, avoiding accidental falls during an odd trajectory waypoint. Their 3D meshes and respective motion will, however, be detected by any visual sensor, yielding images such as shown in Fig. 3.10, and appearing in 3D reconstructions such as the point cloud shown in Fig. 3.6.

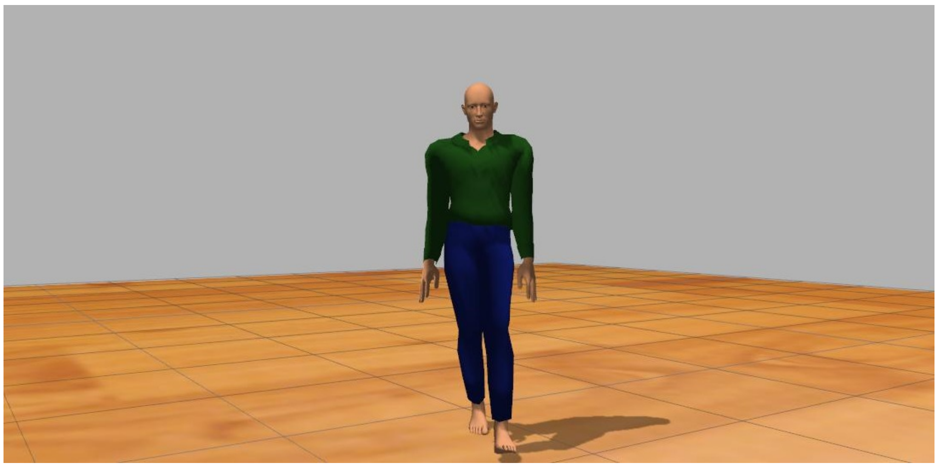


Fig. 3.10 View from a virtual RGB camera of a 3D actor model inserted into an empty Gazebo world.

In a preliminary animated scene of a simple walking skeleton controlling a male 3D model moving in a circular trajectory was implemented, thus simulating a male subject walking in front of the IMPEP set up. This was implemented in the human model XML file itself and then included in the `room_only.world` file, thus building the complete world where the IMPEP will be inserted. More animations will be created in future work taking this preliminary animation as a template, using more complex coding and advanced technologies – this will be discussed in section 4.

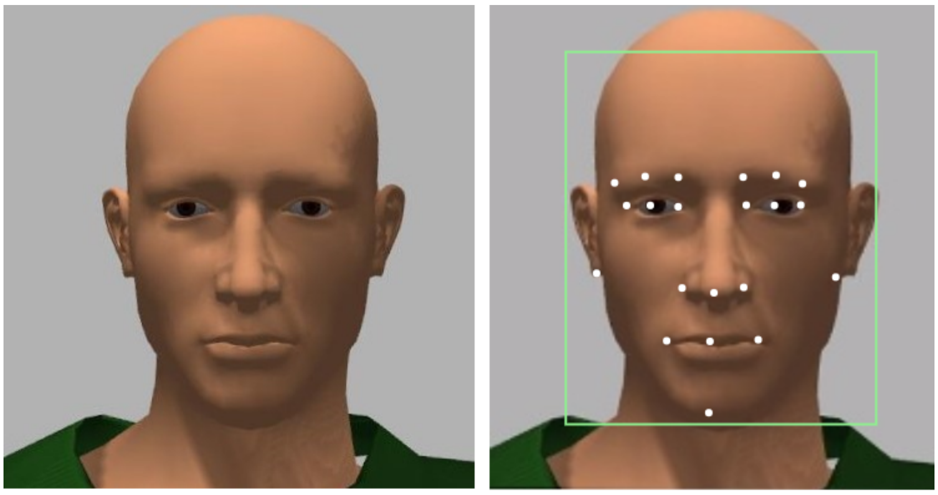
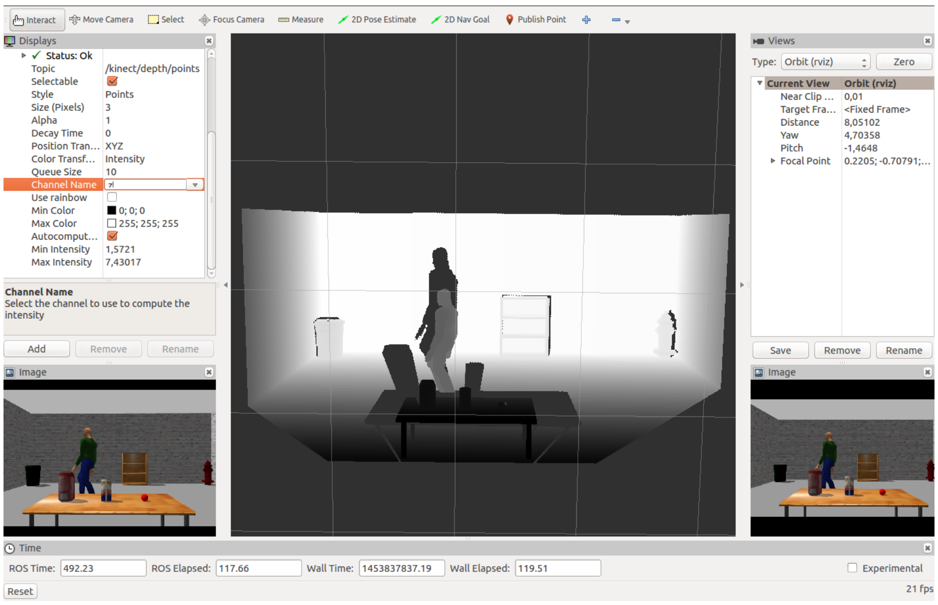


Fig. 3.11 Image of actor model taken by one of the RGB cameras with facial features detected by the CASIR attention middleware (note: in this preliminary test, the image was captured and processed while the actor was stationary).

Experiments in the spirit of the CASIR project, in which artificial attention mechanisms imply both exhibiting attentional behaviour but also reacting to the attentional behaviour of the interlocutor, require that the fixation point of the human subject is determined by the attention framework (a process called *head pose and gaze estimation*). This requires that the 3D model of the actors includes detectable facial features (as seen in figure 3.11) – see [32].

In conclusion, for the purpose of studying the role of attention in HRI, animated human avatars representing experimental subjects will be required to be able to not only perform scripted full-body motion (i.e. walking and/or running, sitting, laying down, etc.), but also scripted head movements, so as to emulate focus-of-attention changes (i.e. gaze shifts).

3.3 Implementation details for the *rqt*-based user interface



Status: Ok	
Topic	/kinect/depth/points
Selectable	✓
Style	Points
Size (Pixels)	3
Alpha	1
Decay Time	0
Position Tran…	XYZ
Color Transf…	Intensity
Queue Size	10
Channel Name	z
Use rainbow	☐
Min Color	■ 0; 0; 0
Max Color	□ 255; 255; 255
Autocomput…	✓
Min Intensity	1,5721
Max Intensity	7,43017

Channel Name
Select the channel to use to compute the intensity

Type: Orbit (rviz) Zero

Current View	Orbit (rviz)
Near Clip …	0,01
Target Fra…	<Fixed Frame>
Distance	8,05102
Yaw	4,70358
Pitch	-1,4648
Focal Point	0.2205; -0.70791;…

ROS Time: 492.23 ROS Elapsed: 117.66 Wall Time: 1453837837.19 Wall Elapsed: 119.51 ☐ Experimental

21 fps

Fig. 3.12 Rviz Interface, visualising simulated camera images and a 3D point cloud reconstruction.

In most projects, spatial visualisation is implemented using *Rviz*, as shown in Fig. 3.12. In this figure, we can see what a GUI would be like in our application context if we only used *Rviz* – in spite of appearing to be a very complete tool, it is not as simple or interactive as required for our project. In order to capitalise on the advantages of *Rviz* while adding increased flexibility in GUI design, *rqt_rviz* was used [33]. This plugin embeds *Rviz* into an *rqt* interface while keeping all of its features and functionalities; however, unlike the *rqt* 2D visualisation plugin, it still has a dependence on its ROS counterpart.

The first step for building an intuitive and simple graphic user interface was to understand the specific needs of the user/developer. With the abundance of visual representations required to monitor camera feeds or processing results from the attention middleware (e.g. point clouds, 3D reconstructions, audio signal waveforms, etc.), the developed GUI must be able to display the greatest variety of information possible, while maintaining an uncluttered dashboard so as to present a maximum level of detail for each data visualisation, and all of this allowing the greatest degree of on-the-fly reconfigurability possible. For development and debugging purposes, the convenience of not having to change windows in the Desktop to access text terminals should be addressed. Therefore, the GUI dashboard was configured so as to allow the display of text terminals in embedded frames in the interface.

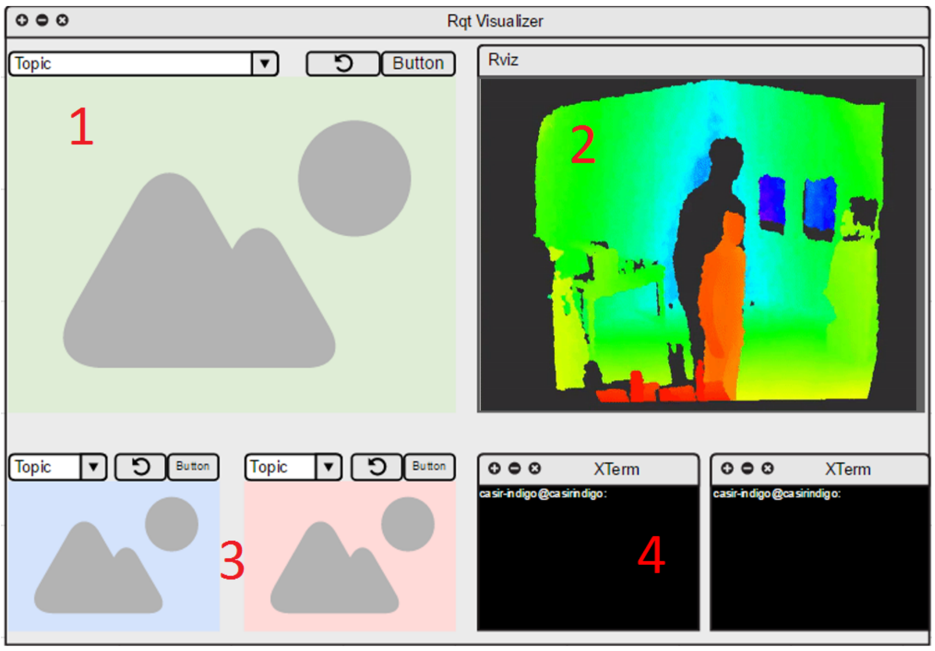


Fig. 3.13 Mockup of the first version of the rqt-based interface. Each frame in the interface in this version, all of which with predefined fixed sizes in the dashboard, was preconfigured to display information as follows: (1) is the main camera visualisation frame; (2) is the main processed data (e.g. 3D reconstructions, signal waveforms, etc.) visualisation frame; (3) contains two secondary frames for camera (or related data, such as depth maps, etc.) visualisation; (4) contains frames for two support terminals. Note, however, that all of these frames are reconfigurable to display whatever is needed by the user/developer on the fly.

A first version of the GUI layout implementing these features is presented in Fig. 3.13. As *rqt* implements the various GUI tools in the form of plugins, there was a need to study which plugins were suitable for this implementation, what is their operating principle and how they could be included in order to comply with this mockup.

The plugin used for 2D visualisation is called *rqt_image_view* [34] – it is an rqt version of ROS's *image_view* [35], in which the system uses *image_transport* to provide classes and nodes capable of transmitting images in arbitrary over-the-wire representations; however they have no dependencies between them. With this plugin, the developer can abstract from the complexity of communication, seeing only *sensor_msg/image* type messages. Alas, *image_view* is not very user friendly, since the desired topic must be selected by specifying it when running the tool in a terminal. Fortunately, the *rqt* version sidesteps this issue by adding a dropdown menu feature showing all of the *sensor_msg/image* messages available. Two additional interesting features of this plugin are save image and topic refresh buttons (relevant in case new publisher nodes are launched).

The final feature of the GUI was the ability to embed a terminal in an interface frame. For this end, we used *rqt_shell* [36], a Python GUI plugin that allows the developer to have several terminals in an *rqt* layout. This plugin supports a fully functional embedded XTerm[3] [36]. The main difference between XTerm and Terminal (Ubuntu's default terminal) is that, despite the fact that the latter includes more features, while XTerm is minimalistic some of its features are more advanced than the ones in Terminal.

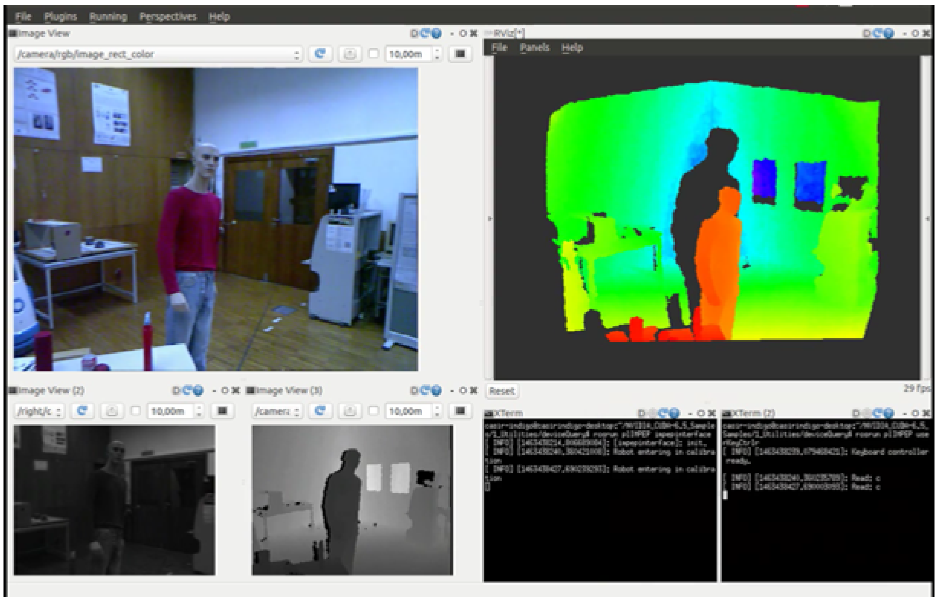
Having chosen the appropriate plugins, these were used to assemble the desired interface. Assembly was performed by resorting to additional interesting features of *rqt*, namely the ability to dock multiple widgets in a single window. Since this feature only defines the relative position of the widgets, in order to fine tune the rqt-based user interface a `.perspective` file needs to be generated and edited to define interface specifications. This file is essentially a configuration file for the layout and all of the plugins – it contains, not only the relative position of every widget, but also the options and configurations of each one of the plugins, such as:

- Default subscribed topics.
- Parameters of Pointcloud representation.
- Terminal default path.
- Rviz active windows and tools.

Using the `.perspective` file, the user can run the *rqt* interface in any computer with a ROS distribution version equal to or above Indigo to be fully functional. We were, therefore, able to meet the important requirement of separating the computational workload resulting from the attentional middleware processing and visualisation, as depicted in Fig. 2.3.

A running instantiation of the first prototype of the GUI is presented in Fig. 3.14.

[3]It therefore needs *xterm* and only works on X11 with Qt 4.



```
casir-indigo@casirindigo-desktop:~/NVIDIA_CUDA-6.5_Samples/1_Utilities/deviceQuery$ rosrun [illegible] [illegible]interface
[ INFO] [illegible]: [[illegible]interface]: init.
[ INFO] [illegible]: Robot entering in calibration
[ INFO] [illegible]: Robot entering in calibration
```

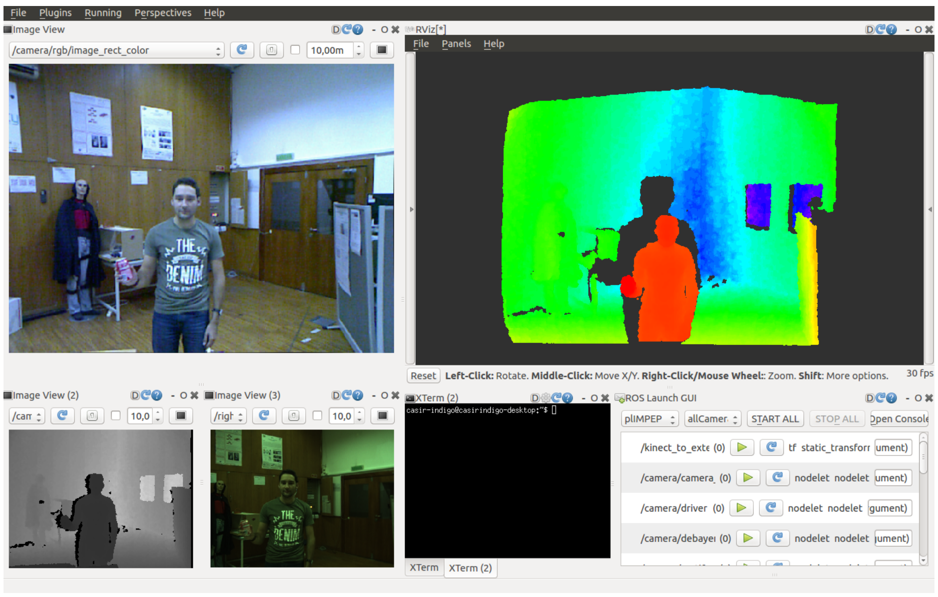
```
casir-indigo@casirindigo-desktop:~/NVIDIA_CUDA-6.5_Samples/1_Utilities/deviceQuery$ rosrun [illegible] [illegible]KeyCtrlr
[ INFO] [illegible]: Keyboard controller ready.
[ INFO] [illegible]: Read: c
[ INFO] [illegible]: Read: c
```

Fig. 3.14 Running instantiation of the first version rqt-based interface applied to the real system. All of the displayed topics are from the real Kinect sensor and RGB cameras – rectified RGB image from left camera on the main frame on the top left, colour-coded 3D point cloud reconstruction from the Kinect on the main frame on the top right, and the grayscale image corresponding to the right camera and a depth map on the secondary frames on the bottom left, respectively from left to right. Instructions such as motor commands or package launching, and also other system commands, are conveniently conveyed through the two embedded terminals on the bottom right frames.

Despite this first prototype already being a relatively complete and functional user interface, it is more appropriate for developers who are knowledgeable enough to run necessary launch files through the terminals (e.g. to switch between feeds of the real system to the outputs of the Gazebo simulator, or to run or stop any specific set of attention middleware modules), so there was a need to further improve the interface by adding a new feature – a user-friendly package launcher. To do so we explored an experimental plugin named *rqt_launch*, which allows the user to:

- See all `.launch` files on the local file system through a dropdown menu.
- See all nodes defined in a `.launch` file (after selecting a `.launch` from the previous item).
- Run and stop the active `.launch` file.
- Run and stop individual nodes from the active `.launch` file

Consequently, this feature was added together with an improvement to the terminal plugin, allowing it to display two windows in the same space (with the use of tabs). The final version of the *rqt*-based interface is shown in Fig. 3.15.



```
casir-indigo@casirindigo-desktop:~$
```

Fig. 3.15 Instantiation of the final version rqt-based interface applied to the real system. All of the topics are from real IMPEP sensors, as in Fig. 3.14. On the bottom right frame, the launcher plugin can be seen in action – files and nodes are launched using interface buttons provided by this plugin.

3.4 Implementation details for the web service supporting the CASIR-IMPEP remote lab

The second goal of the work presented in this thesis, as stated in section 1.1, was to provide a means for any authorised researcher from anywhere outside the laboratory to access the CASIR-IMPEP framework via a common web browser. As stated in section 2.5, this module was designed so as to use a client-server architecture.

As explained in section 2.5, to open a ROS environment to non-ROS systems, *Rosbridge* is the tool of choice. Additionally, since streams of image topics are to be displayed in the HTML interface, therefore requiring a sustained connection with the appropriate bandwidth and upload/download speeds, the *Web_Video_Server* tool was also used [37].

The first implementation step was to set up the server side. *Rosbridge* is a package that does not come in the native ROS installation, so it needs to be installed using: `sudo apt-get install ros-<rosdistro>-rosbridge-suite`, with `<rosdistro>` corresponding to the

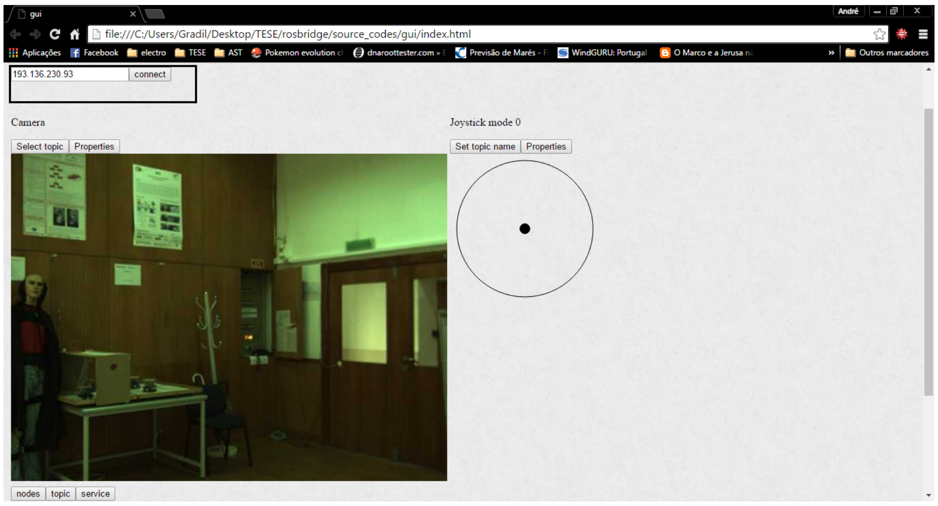


Fig. 3.16 CASIR-IMPEP remote lab, the HTML web page already connected to the server and streaming one topic.

appropriate ROS distribution, in our case Indigo. After installing, the server IP and port need to be configured. As the laboratory has a firewalled LAN, a “tunnel” had to be created in order to grant outside access to the main project computer (that will be our server).

After the connection was configured, it was necessary to create and configure the video stream as well using *Web_Video_Server*. This tool opens a local port, and waits for incoming HTTP requests. When a video stream of a ROS image topic is requested via HTTP, it subscribes to the corresponding topic and creates an instance of the video encoder.

In order to easily launch all the packages necessary to run the servers and middleware launch files, several shell scripts were created. The main shell script is responsible for launching:

- A roscore node.
- Rosbridge Server.
- Web Video Server.
- All of the real IMPEP’s sensors.

There is also a simulation counterpart of this script, that launches the Gazebo simulator instead. Any desired combination of launches can be achieved and scripted according to specific needs. The launches, however, don’t necessarily need to be scripted: they can be ran individually through the terminal by the user.

For the client side of the web service, the user interface was created using a simple HTML file – see Fig. 3.16. The created webpage appears to its user as an empty frame with a connection button and a textbox with a default IP address (which should correspond to the server IP address). In a lower layer, however, it has JavaScript modules that communicate with Rosbridge through websockets, namely:

- `Starting_01.js` is responsible for loading the page and initializing an empty connection.
- `Connection_01.js` creates the connection defining the IP address, advertises, unadvertises, subscribes, unsubscribes topics and services.
- *Ros.js* ensures that JSON and WebSocket are available. An exception is thrown if not.
- `Info_01.js` gives information about topics, nodes and services running at server.
- `Camera_01.js` initialises the camera GUI and button listeners.
- `Joystick_01.js` sends control signals to defined command topics by joystick.

To create these scripts, we used as a template the work by Blaha et al., 2013 [38], updated them with contemporary plugins and custom parameters so as to conform with our requirements.

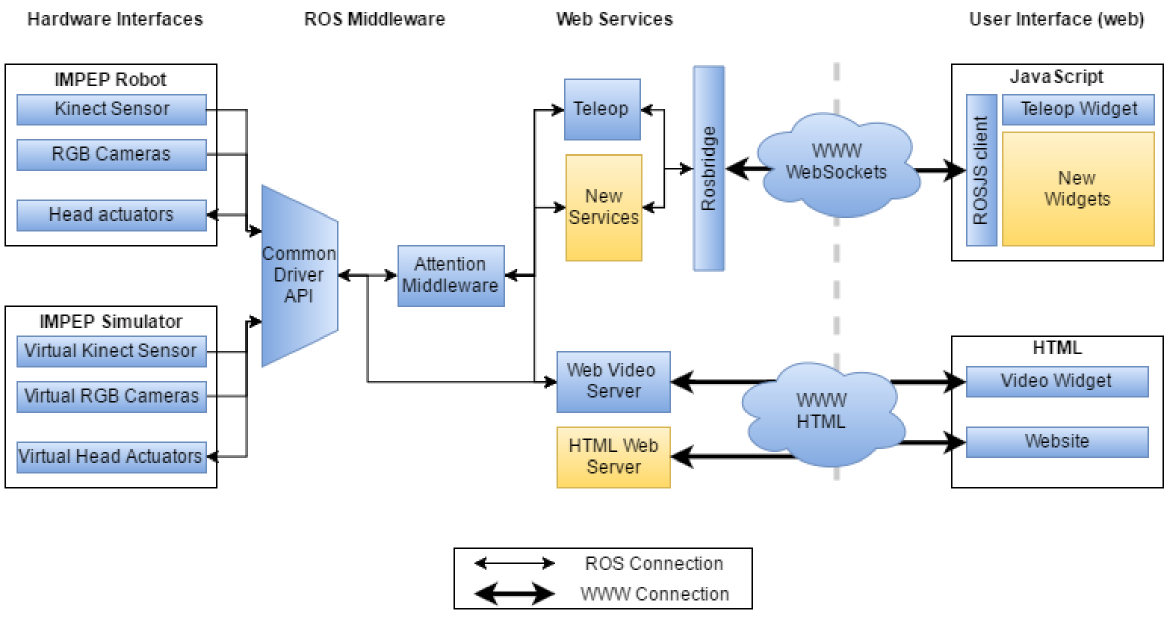


Fig. 3.17 Complete dataflow diagram of the web service. It can be divided into four layers: Hardware Interfaces, ROS Middleware, Web Services and finally User Interface. The modules in yellow represent the possibility to expand the system with new features. The "HTML Web Server" represents an online host for the website, at present time the can be used with direct possession of the html file and javascripts.

An overview of the complete system, from server to client passing through the communication protocols, can be seen in Fig. 3.17, including possibilities of expansion[4].

3.5 Experimental results

The final stage of the work presented in this thesis was to test and evaluate the performance of the developed framework in full-blown operation. To better understand the conditions under which these tests were conducted, in the following text the specifications of the computational hardware supporting the framework (Fig. 2.3) will be presented in detail.

The **main computer** comprises:

- GPU: 2 Asus GTX780 3GB DirectCU II OC units.
- CPU: 4 core Intel® Core™ i7-3770 @ 3.40GHz.
- Motherboard: Asrock Z77 OC Formula.
- RAM: 16GB DDR3-1600Mhz.
- SSD: 128GB KINGSTON SSDNow V200.
- HDD: Seagate Barracuda 1TB.

The **visualisation computer** is expected not to be as powerful as the main unit. To validate this assumption we used a virtual machine (emulated in VMWare software) with the following specifications:

- GPU: Gallium 0.4 on SVGA3D.
- CPU: 2 core Intel® Core™ i5-6300HQ CPU @ 2.30GHz.
- RAM: 3GB DDR3-2300MHz.
- HDD: 76.0 GB.

As for the remote CASIR-IMPEP lab clients, several different machines with various operating systems were used for testing.

Our solution was evaluated in terms of features, usability and technical benchmarks – results of this evaluation will be described in the following sections.

[4]More details in section 4

3.5.1 Simulation and visualisation proof-of-concept

The CASIR-IMPEP simulator was designed, as explained in previous text, to provide a virtual IMPEP model capable of interacting with the artificial attention middleware through ROS, to provide a virtual representation of the laboratory, and finally to offer an animated human model with enough features to be used in conjunction with the CASIR middleware so as to produce functioning simulated HRI experiments.

These features were successfully implemented:

- The robotic model was created and is fully functional. It was compared manually to the real IMPEP in terms of movement – while on manual control, sending the commands to both versions of IMPEP at the same time, they had the same behaviour and response time.
- The experimental environment was implemented with enough detail (i.e. objects, table, etc.) so as to reproduce the most important aspects of the work area for HRI described in [4], as seen in Fig. 3.9.
- A human model was added to the environment and its features proved to be realistic enough for appropriate processing by the attention middleware, as can be seen in Fig. 3.18.

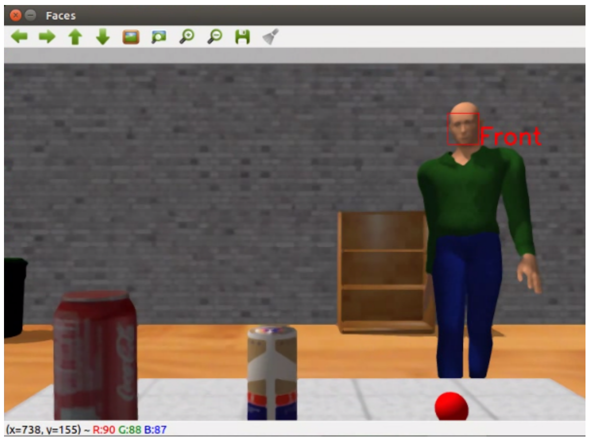


Fig. 3.18 Face and pose detection in simulated environment seen through a post-processed output window.

Fig. 3.19 shows a global view of the simulator taken from a third person perspective with a high-resolution virtual camera. This third person perspective, besides providing a useful overview of the current state of a given experiment, also allows conducting important technical benchmarks, in order to assess CPU and GPU loads as well as RAM usage in the main computer.

CPU and RAM usage was measured using *htop*[5], while GPU usage was measured using *nvidia-smi*[6].

With further configuration of the simulator launch file, we were able to launch only the backend of Gazebo – in other words, the Gazebo server without the client (Desktop UI). This way all of the packages, controllers, computations and topics were active and being published in ROS, however only in background. As a consequence, the user had no way of seeing what is happening in the simulated world other than through IMPEP's cameras and the third-person view, making it all the more useful the existence of the latter for monitoring the simulated experiment.

Table 3.4 Gazebo UI *vs* without Gazebo UI benchmark comparison. Percentages and memory usage are relative to the specifications of the main computer

	CPU	**RAM**	**GPU**
UI	24.18%	1422Mb	438Mb
Without UI	12.51%	1267Mb	276Mb

As can be seen in Table 3.4, the absence of a Gazebo UI reduced almost 12% of the CPU load despite the need for an extra camera with higher resolution. As for main memory usage, the difference was not that remarkable. GPU usage, on the other hand, was expected to also drop slightly, since there was no need to render the entire world in real-time.

[5]htop is an interactive process viewer for Unix systems. It is a text-mode application (for console or X terminals).

[6]nvidia-smi is a command line utility, supported by the NVIDIA Management Library (NVML), intended to aid in the management and monitoring of NVIDIA GPU devices.

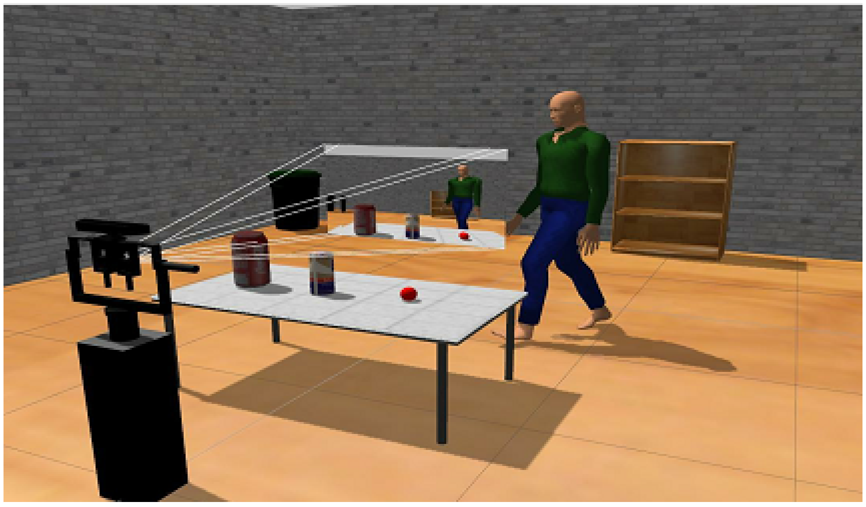
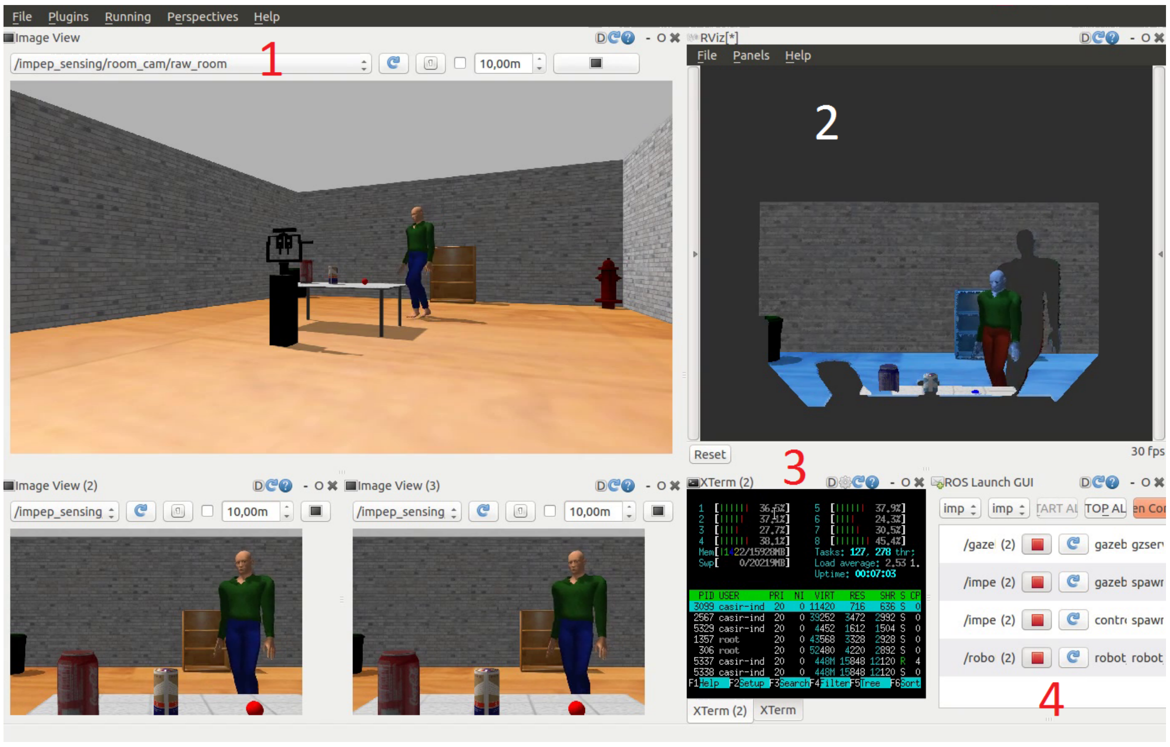


Fig. 3.19 Final simulated environment in action with human model walking, seen from the third-person perspective of the external simulated high-resolution camera.

The CASIR-IMPEP GUI was designed, as explained previously, to be fully connected with the ROS middleware, to provide an intuitive interface with which any user can easily select any topics he/she wants to visualise, namely complex views of processed data, and finally to offer a means to start or stop any desired set of system modules. As a consequence:

- All of the *sensor_msgs/Image* messages published in ROS are visible in the UI.
- As seen in Fig. 3.20, in most of the frames in the dashboard, the user only needs to select the desired topic from drop-down menus.
- Visualisation of complex displays of processed data has been implemented. As an example, Fig. 3.20 shows us, in frame (2), the *Rviz* plugin displaying the pointcloud data.
- The user can start and stop launch files, either using terminal commands in (3) or from the launcher in (4) – see figure 3.20.



```
  1  [||||||    36.5%]     5  [||||||    37.9%]
  2  [|||||     37.1%]     6  [||||      24.3%]
  3  [||||      27.7%]     7  [|||||     30.5%]
  4  [||||||    38.1%]     8  [||||||||  45.4%]
  Mem[||1422/15928MB]      Tasks: 127, 278 thr;
  Swp[     0/20219MB]      Load average: 2.53 1.
                           Uptime: 00:07:03

  PID USER       PRI  NI  VIRT   RES   SHR S CP
 3099 casir-ind   20   0 11420   716   636 S  0
 2567 casir-ind   20   0 39252  3472  2992 S  0
 5329 casir-ind   20   0  4452  1612  1504 S  0
 1357 root        20   0 43568  3328  2928 S  0
  306 root        20   0 52480  4220  2892 S  0
 5337 casir-ind   20   0  448M 15848 12120 R  4
 5338 casir-ind   20   0  448M 15848 12120 S  0
F1Help  F2Setup F3SearchF4FilterF5Tree  F6Sort
```

Fig. 3.20 [Final rqt interface with simulator topics and human model walking seen from high resolution room camera (1). Depth information seen in *rviz*(2), cpu and memory usage in the embedded terminal (3) and also the launcher (4).

Exhaustive tests were conducted to evaluate visualisation performance, namely including CPU, RAM and GPU measurements, either running the GUI directly in the **main computer** or passing topics to the **visualisation computer**, where they were shown using the rqt interface running in a local ROS installation. These tests serve the purpose of assessing how much of an added value there is in running visualisation in a separate computer.

More specifically, two operational scenarios were used: (1) **local visualisation**, where we use the main computer to run everything, including real sensor drivers management and the visualisation of IMPEP camera topics and point cloud data; (2) **external visualisation**, where the main computer still runs real sensor drivers, but all visualisation is relegated to the separate dedicated computer, described above, displaying the same topics.

Table 3.5 Local vs remote visualisation benchmark comparison, the specifications of both machines are those in section 3.4

Main Computer Benchmark			
	CPU	**RAM**	**GPU**
Local Visualisation	12.51%	1444Mb	344MB
External Visualisation	9.00%	1038Mb	144MB
Visualisation Computer Benchmark			
Local Visualisation (visualisation computer idle)	2%	824MB	N/A
External Visualisation	61.30%	1217MB	N/A

Table 3.5 shows the results for the two operational scenarios, in which we can see that in the **external visualisation** scenario, the CPU load decreases in 3.51% for the main computer. Furthermore, a decrease 400MB of RAM used and 200MB of graphical memory in this computer is observed when comparing with the **local visualisation** scenario. On the other hand, analysing benchmark values for the visualisation computer, we can see that its computational load increases – CPU usage suffers by far the largest increase; however, we need to take into consideration that the percentage for the main computer referes to a 4 core, 8-threaded CPU with 3.4 GHz of clock frequency, while the visualisation computer only has 2 cores and 2 threads of lower clock frequency for the exact same computations. Conversely, memory usage has a more direct comparison since it increases exactly in the amount it decreases for the main computer.

Finally, we can measure the computational weight of both simulation and visualisation running simultaneously, comparing the effects of the lack of UI.

Table 3.6 Results of the combining the aforementioned tests – data from the main computer.

	CPU	**RAM**	**GPU**
UI + VIS	49.65%	2048Mb	535Mb
no UI + VIS	35.48%	1603Mb	368Mb
no UI + Remote VIS	31.38%	1240Mb	180Mb

Taking a closer look at Table 3.6, data is still coherent with the previous results: CPU drops significantly after taking out the UI and even further with remote visualisation. As for memory usage, conclusions are similar.

Testing these modules while connecting them to *allnodes.launch*, the main launch file that runs every node of the attention middleware, however, becomes problematic.

Table 3.7 Results of the simulation and visualisation with the full attention middleware running, data from the main computer.

	CPU	**RAM**	**GPU**
Allnodes	99.93%	2956Mb	1662Mb
Allnodes + VIS	100.00%	3088Mb	1717Mb
Allnodes + VIS + no UI	100.00%	3310Mb	1879Mb

Having tested the computational burden of this launch alone we obtained the results presented in Table 3.7. It is easy to see that the major system load comes from the middleware itself, from **Allnodes + VIS** to **Allnodes + VIS + no UI** there is no difference in CPU usage because it is already overloaded, the noticeable difference in this case is the decrease of the real time factor in the simulator, it decreases to about 0.6.

In other words, using the current setup, and even relegating visualisation to an external computer and removing the need to fully render the simulated environment on Gazebo, each second passed in the "real world", only 0.6 seconds were processed in the simulator, a very important factor that needs to be taken into account by future developers and users.

3.5.2 Remote simulation and control proof-of-concept

The remote lab is the module for which less features were implemented and with more room for improvement. Currently, only the core web service and a front-end interface of the remote lab have been implemented. The interface, following the diagram of Fig. 3.17, allows the user to see topics from the main computer from anywhere with just an internet connection and the HTML file itself. It also allows to send commands to defined topics through a joystick feature.

In summary, using the remote lab currently a user can:

- Select what topics to visualise.
- See what nodes, topics and services are available and running.
- Change the resolution and/or quality of the stream, to adjust to the speed of the internet connection of the client.
- Control the simulated version of IMPEP with a joystick.

The developer, someone with the capability of editing the JavaScripts above mentioned, on the other hand, can do more than that, since the HTML and scripts are not hosted online at the moment:

- Change the encryption of the video stream.
- Change any default values and parameters of the scripts.
- Create and include new scripts with ease - as the web site was created in a modular fashion.

In order to benchmark network resource usage, the remote lab was tested through three separate internet connections, specified in Table 3.8. In this study, several influential experimental conditions were fixed, such as selected topic `/right/camera/image_rect_color` (real system), resolution (640x480 the camera's default), stream quality at 100%. Additionally, in all experimental runs the chosen browser was Google Chrome (the most optimized for *Web_video_server* applications[7].)

Table 3.8 Internet connection benchmark. 1 - cabled connection inside the lab, 2 - wifi connection at the Electrical Engineering Department (same building), 3 - wifi connection at home.

	Download	**Upload**	**Ping**	**Fps (avg.)**	**Bandwidth (avg.)**
Ethernet Cable	79.61 Mbs	96.12 Mbs	5 ms	25	12.2 Mbs
Wireless 1	23.90 Mbs	23.70 Mbs	5 ms	25	12.2 Mbs
Wireless 2	15.54 Mbs	17.47 Mbs	6 ms	25	12.2 Mbs

Considering the specifications of the three connections, and assessing their performance in the conditions, we can confirm that the system is stable, exhibiting an fps average of 25 and 12.2Mbs of bandwidth occupation. Note that the same test was repeated having both wireless clients running at the same time, without any change of values in each case.

The same test was performed with the simulated environment, and in this case, mostly due to including joystick publishing commands (manual remote command of the real IMPEP head is not allowed, for obvious safety reasons), average fps dropped to 22 and the bandwidth occupation increased to 16Mbs. In any case, this does not represent a significant change in performance.

[7]in point 3-Latency of [37]

Chapter 4

Conclusions and future work

Throughout this text, and in particular in Chapter 3, the overall objective and goals defined in section 1.1 were shown to be satisfactory attained, and all the identified requirements met.

The simulator offers a faithful and useful model of IMPEP, and also provides a functional virtual environment for HRI experimentation, including a simulated human actor with animation capabilities and enough features for the CASIR attention middleware to promote purposeful interaction.

The *rqt*-based visualisation software is intuitive, exhibits 2D and 3D visualisation capabilities, features easy ROS topic selection, a launcher selector and advanced user terminal embedding for debug and/or manual launch. It can also be used in a separate workstation to save precious computational resources, and, as demonstrated in section 3.5.1, it can be successfully ran in even a very low end computer with few restrictions.

As for the remote robotic laboratory it can connect researchers outside of the lab to the framework via the Internet, and can even provide control of the simulated platform motors.

Every component acts in a synergistic fashion so as to complement the base framework in a simple, modular, and optimised fashion, trying in full to minimise resource usage and computational burden of the main computer.

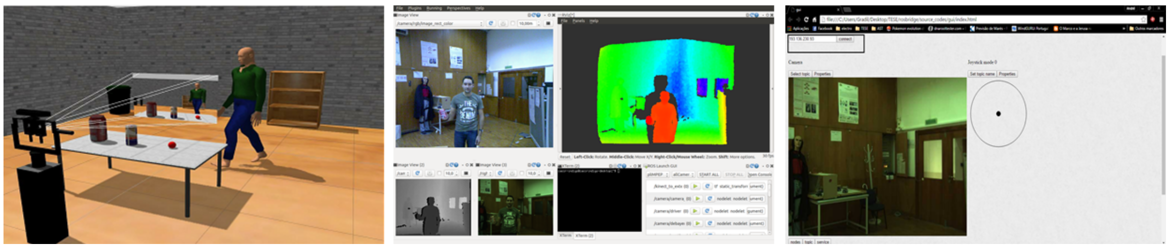


Fig. 4.1 Overview of the several components of the proposed solution running in operational conditions.

Several improvements can be added to the proposed framework in future work. Further improvements have been planned in two tracks: software improvement and hardware inclusion.

In the first case – software improvement – starting with the Gazebo simulator, the major advance can be made in terms of furthering the development of the simulated human actors. The model itself can be improved (despite already displaying core features), for example by resorting to 3D scanning [39] and creating the skeleton and joints accordingly, making it possible to have a much more realistic reproduction of a human subject within the simulator. In fact, on a much larger scale this can be applied to the entire room. A second improvement related with the human model consists in the creation of a suite of action scripts, and later on the development of a user-friendly toolkit to assist with script generation. Finally, it would be important to add more joints to the skeleton, so as to allow for more sophisticated simulation of human-robot interaction details, such as gaze shifts.

As for the *rqt* visualiser, the launcher section of the UI could be redesigned in order to make it even more user-friendly, especially on what concerns non-developers. An improvement that has already been planned for the near future involves implementing the layout sketched in Fig. 4.2.

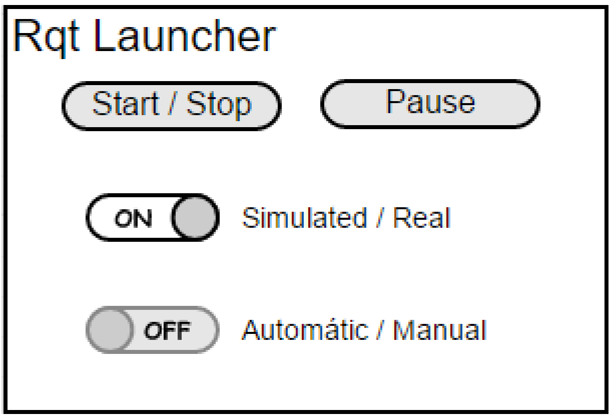


Fig. 4.2 New launcher plugin interface.

With this simplified design, two buttons and two switches, the main launch sequences could be performed in one go, giving an advanced user still a way to launch more specific nodes through the embedded terminals already implemented, or even through alternative button dashboards. This new layout would allow to launch the following combinations:

- Simulation environment with all attention nodes (except the real cameras) .
- Simulation environment with manual motor position control.

- Real system with all attention nodes.
- Real system with manual motor control nodes.

Other obvious improvements would be to develop visualisation screens for the diverse data displays needed for attention middleware development and monitoring (as mentioned before, for example, different types of 3D reconstructions, signal waveforms, etc.).

Finally, as for the remote lab, the most important follow-up work would be, hosting the web site in a server and to provide a safe handshake procedure for access authorisation. This will imply the creation of a welcoming homepage through which the users would login and request time slots to the webmasters, as currently only the action page of the remote robotic lab has been developed. The action page should only be accessible to users already accepted by the webmasters as the current time slot owners. In order to implement this, a PHP (PHP: Hypertext Preprocessor) and SQL (Structured Query Language) front-end is needed. This will allow the creation of a user database and a login system, essential for the future usability and true opening to the outside community while ensuring safe access.

In terms of hardware inclusion, as stated in section3.2, Gazebo 4 comes with Oculus Rift and Razer Hydra compatibility, which would give the user a higher level of immersion, and more possibilities of human/object interaction inside the simulator.

References

[1] International Federation of Robotics, "Statistics - IFR International Federation of Robotics." [http://www.ifr.org/industrial-robots/statistics/ Online; accessed 1-August-2016].

[2] V. Tikhanoff, A. Cangelosi, P. Fitzpatrick, G. Metta, L. Natale, and F. Nori, "An OpenSource Simulator for Cognitive Robotics Research The Prototype of the iCub Humanoid Robot Simulator." 2008.

[3] L. Gomes and S. Bogosyan, "Current trends in remote laboratories," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 12, pp. 4744–4756, 2009.

[4] P. Lanillos, J. N. Oliveira, and J. F. Ferreira, "Experimental Setup and Configuration for Joint Attention in CASIR," Tech. Rep. MRL-CASIR-2013-11-TR001, Mobile Robotics Lab – Institute of Systems and Robotics, Coimbra, Portugal, November 2013.

[5] F. Weißhardt, "Care-o-bot-research: Providing robust robotics hardware to an open source community," 2011. [presentation].

[6] G. Metta, P. Fitzpatrick, and L. Natale, "YARP: Yet Another Robot Platform," *International Journal on Advanced Robotics Systems*, p. 3(1):43–48, 2006.

[7] T. Construct, "The Construct - Just Simulate!," 2016. [http://www.theconstructsim.com/ Online; accessed 29-February-2016].

[8] J. F. Ferreira and J. Dias, "Attentional Mechanisms for Socially Interactive Robots–A Survey," in *IEEE Transactions on Autonomous Mental Development*, vol. 6, pp. 110 – 125, IEEE, 2014.

[9] P. Lanillos, J. F. Ferreira, and J. Dias, "Designing an Artificial Attention System for Social Interaction," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4171 – 4178, IEEE, 2015.

[10] P. Lanillos and J. F. Ferreira, "The CASIR-IMPEP Attention Framework for Social Interaction with Robots," Tech. Rep. MRL-CASIR-2013-12-TR004, Mobile Robotics Lab – Institute of Systems and Robotics, Coimbra, Portugal, December 2013.

[11] M. Quigley, K. Conley, B. 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*, vol. 3, p. 5, Kobe, Japan, 2009.

[12] S. Ivaldi, J. Peters, V. Padois, and F. Nori, “Tools for simulating humanoid robot dynamics: A survey based on user feedback,” in *2014 IEEE-RAS International Conference on Humanoid Robots*, pp. 842 – 849, IEEE, 2014.

[13] Open Source Robotics Foundation, “Gazebo,” 2014. [http://www.gazebosim.org/ Online; accessed 12-July-2016].

[14] LAAS-CNRS, “MORSE.” [https://www.openrobots.org/morse/doc/stable/morse.html Online; accessed 12-July-2016].

[15] Coppelia Robotics, “V-Rep: Virtual Robot Experimentation Platform.” [http://www.coppeliarobotics.com/ Online; accessed 12-July-2016].

[16] Cyberbotics, “Webots - robot simulator.” [https://www.cyberbotics.com/ Online; accessed 12-July-2016].

[17] R. Diankov, R. Ueda, K. Okada, and H. Saito, “COLLADA: An Open Standard for Robot File Formats,” in *2011 RSJ The 29th Annual Conference of the Robotics Society of Japan*, 2011.

[18] OSRF, “srs_ui_pro - ROS wiki.” [http://wiki.ros.org/srs_ui_pro Online; accessed 12-July-2016, last edit 17-October-2012 10:54:10].

[19] The Qt Company, “Qt.” [https://www.qt.io/developers/ Online; accessed 09-August-2016].

[20] wxWidgets, “Wxwidgets: Cross-Platform GUI Library.” [http://www.wxwidgets.org/ Online; accessed 09-August-2016].

[21] OSRF, “Rqt - Ros Wiki .” [http://wiki.ros.org/rqt Online; accessed 09-August-2016, last edit 11-February-2015 18:57:04].

[22] A. L. Taylor and J. T. Wright, “A telerobot on the world wide web,” in *In National Conference of the Australian Robot Association*, 1995.

[23] C. Crick, G. Jay, B. Pitzer, and O. C. Jenkins, “Rosbridge: Ros for non-ros users,”

[24] B. Pitzer, S. Osentoski, G. Jay, C. Crick, and O. C. Jenkins, “Pr2 remote lab: An environment for remote development and experimentation,” in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 3200–3205, IEEE, 2012.

[25] “PR2 Remote Lab.” [https://sites.google.com/site/sosentos/Home/research Online; accessed 24-August-2016].

[26] R. Toris, J. Kammerl, D. V. Lu, J. Lee, O. C. Jenkins, S. Osentoski, M. Wills, and S. Chernova, “Robot web tools: Efficient messaging for cloud robotics,” in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pp. 4530–4537, IEEE, 2015.

[27] J. Bankoski, P. Wilkins, and Y. Xu, “Technical overview of vp8, an open source video codec for the web.,”

[28] J. F. Ferreira, J. Lobo, and J. Dias, “Fast Exact Bayesian Inference for High-Dimensional Models,” in *Workshop on Unconventional Computing for Bayesian Inference, 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2015)*, 2015.

[29] Allied Vision Technologies GmbH, “GUPPY F-036 Datasheet.” [https://www.alliedvision.com/en/products/cameras/detail/Guppy/F-036.html Online; accessed 24-August-2016].

[30] Harmonic Drive AG, “Engineering Data DC servo Actuators PMA.” [http://harmonicdrive.de/mage/media/catalog/category/ED_PMA_E_1019821_12_2015_V01_6.pdf Online; accessed 24-August-2016].

[31] M. Kuniecki, J. Pilarczyk, and S. Wichary, “The color red attracts attention in an emotional context. an erp study,” *Frontiers in human neuroscience*, vol. 9, 2015.

[32] E. Murphy-Chutorian and M. M. Trivedi, “Head pose estimation in computer vision: A survey,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 31, no. 4, pp. 607–626, 2009.

[33] OSRF, “Rqt_rviz - Ros Wiki .” [http://wiki.ros.org/rqt_rviz Online; accessed 09-August-2016, last edit 06-September-2012 18:45:49].

[34] OSRF, “Rqt_image_view - Ros Wiki .” [http://wiki.ros.org/rqt_image_view Online; accessed 09-August-2016, last edit 25-November-2015 18:56:46].

[35] OSRF, “Image_view - Ros Wiki .” [http://wiki.ros.org/image_view Online; accessed 09-August-2016, last edit 18-July-2016 20:32:06].

[36] OSRF, “Rqt_shell - Ros Wiki .” [http://wiki.ros.org/rqt_shell Online; accessed 09-August-2016, last edit 02-May-2013 18:45:49].

[37] OSRF, “Web_video_server - Ros Wiki .” [http://wiki.ros.org/web_video_server Online; accessed 09-August-2016, last edit 09-March-2015 05:49:16].

[38] M. Blaha, M. Krec, P. Marek, T. Nouza, and T. Lejsek, “Rosbridge web interface,” tech. rep., Department of Cybernetics Faculty of Electrical Engineering, Czech Technical University Technická, 166 27 Prague 6, Czech Republic, May 2013.

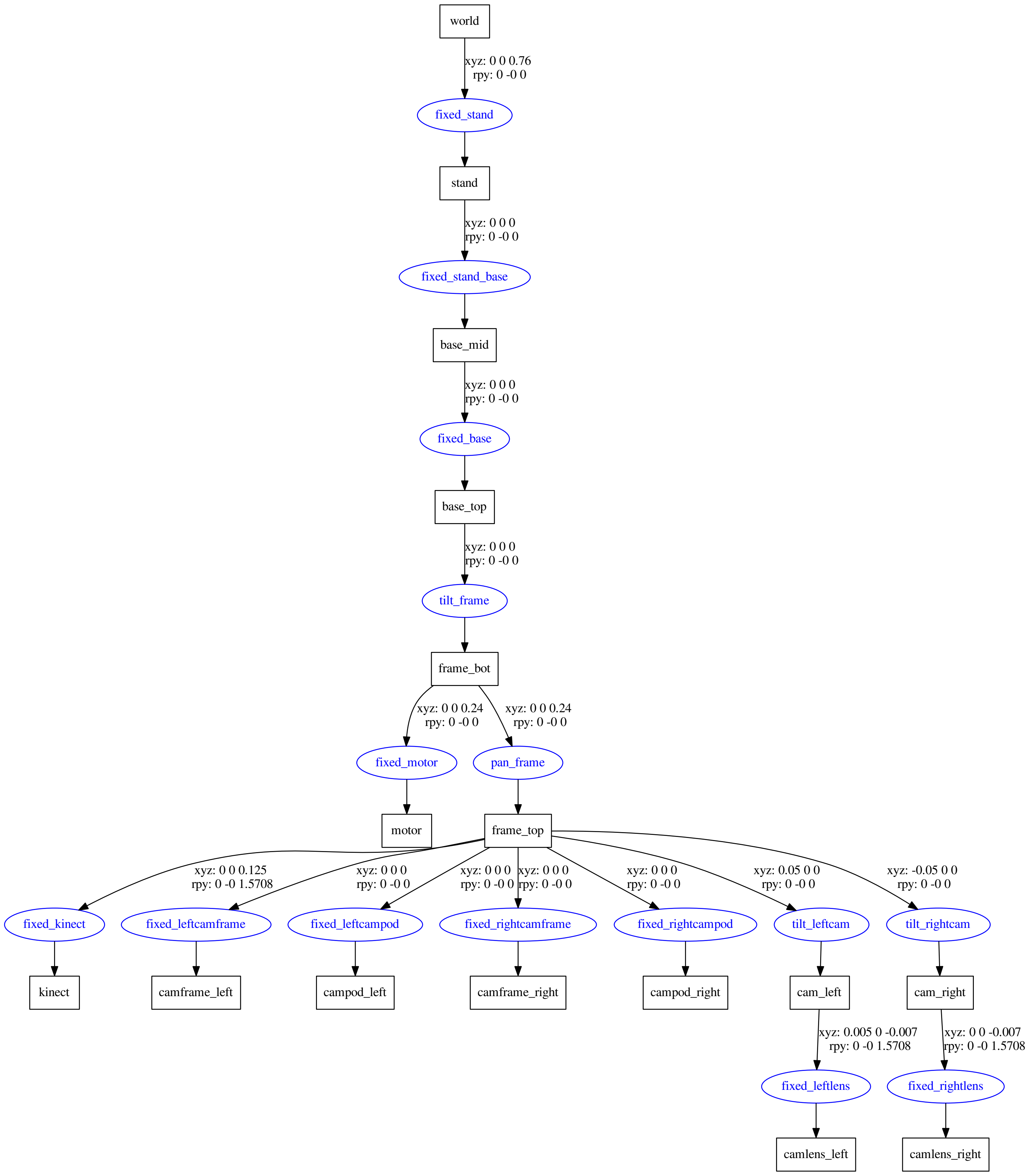
[39] J. Tong, J. Zhou, L. Liu, Z. Pan, and H. Yan, “Scanning 3d full human bodies using kinects,” *IEEE transactions on visualization and computer graphics*, vol. 18, no. 4, pp. 643–650, 2012.

Appendix A

Parsed URDF kinematic chain

```
//------------------------ stand-world joint ------------------------//
  <link name='stand'>
    <inertial>
      <mass value='500'/>
      <origin xyz='0 0 .25' rpy='1.57079633 0 0'/>
      <inertia ixx="1" ixy="0" ixz="0" iyy="1" iyz="0" izz="1"/>
    </inertial>

    <visual name='stand_visual'>
      <origin xyz='0 0 .25' rpy='1.57079633 0 0'/>
      <geometry><mesh filename = 'model://impep/meshes/stand.dae'/></
   geometry>
    </visual>

    <collision name="stand_collision">
      <origin xyz='0 0 .25' rpy='1.57079633 0 0'/>
      <geometry><mesh filename = 'model://impep/meshes/stand.dae'/></
   geometry>
    </collision>
  </link>

  <link name="world"/>

  <joint name="fixed_stand" type="fixed">
    <origin xyz="0 0 0.76" rpy="0 0 3.1415"/> <!-- a ver-->
    <parent link="world"/>
    <child link="stand"/>
  </joint>
```

Listing A.1 Code graft regarding the first link of the model and the world itself


world
xyz: 0 0 0.76
rpy: 0 -0 0
fixed_stand
stand
xyz: 0 0 0
rpy: 0 -0 0
fixed_stand_base
base_mid
xyz: 0 0 0
rpy: 0 -0 0
fixed_base
base_top
xyz: 0 0 0
rpy: 0 -0 0
tilt_frame
frame_bot
xyz: 0 0 0.24
rpy: 0 -0 0
xyz: 0 0 0.24
rpy: 0 -0 0
fixed_motor
pan_frame
motor
frame_top
xyz: 0 0 0.125
rpy: 0 -0 1.5708
xyz: 0 0 0
rpy: 0 -0 0
xyz: 0 0 0
rpy: 0 -0 0
xyz: 0 0 0
rpy: 0 -0 0
xyz: 0 0 0
rpy: 0 -0 0
xyz: 0.05 0 0
rpy: 0 -0 0
xyz: -0.05 0 0
rpy: 0 -0 0
fixed_kinect
fixed_leftcamframe
fixed_leftcampod
fixed_rightcamframe
fixed_rightcampod
tilt_leftcam
tilt_rightcam
kinect
camframe_left
campod_left
camframe_right
campod_right
cam_left
cam_right
xyz: 0.005 0 -0.007
rpy: 0 -0 1.5708
xyz: 0 0 -0.007
rpy: 0 -0 1.5708
fixed_leftlens
fixed_rightlens
camlens_left
camlens_right