A Visualisation and Simulation Framework for Local and Remote HRI Experimentation

André Gradil and João Filipe Ferreira
Institute of Systems and Robotics (ISR)
Dept. of Electrical & Computer Eng.
University of Coimbra
Pinhal de Marrocos, Polo II
3030-290 COIMBRA, Portugal

***Abstract*—In this text, we will present work on the design and development of a ROS-based (Robot Operating System[1]) 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 (Coordinated Attention for Social Interaction with Robots) project. The proposed solution was implemented using ROS, and designed to attend the needs of two user groups – local and remote users and developers. The framework consists of: (1) a fully functional simulator integrated with the ROS environment, including a faithful representation of a robotic platform, 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 for monitoring and development; (3) a remote robotic laboratory that can connect remote users to the framework via a web browser. 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. Finally, conclusions concerning the success and potential of this research and development effort are drawn, and the foundations for future work will be proposed.**

***Index Terms*—Visualisation, Simulation, Remote, User Interface, ROS, Gazebo, Framework.**

I. Introduction

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

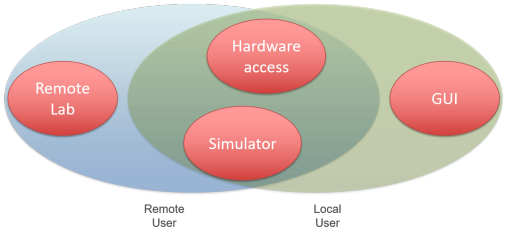

Fig. 1: Desired features for most contemporary robotic development frameworks.

The combined set of desired features resulting from this demand and its relationship with potential user types is depicted in Fig. 1. The overall objective of the work presented in this text 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[2], with these features, as a follow-up on future work planned in [3] see Fig. 2. This system is supported by the IMPEP infrastructure (acronym for Integrated Multimodal Perception Experimental platform) – see Fig. 3[3] More specifically, the work presented in this text had the following main goals: (1) the development of IMPEP hardware and simulator access to local users, with support of a intuitive local GUI; (2) providing

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

[2]FCT Contract PTDC/EEI-AUT/3010/2012, which ran from 15-04-2013 until 31-07-2015. The motivations for this work can be found in [4], while conceptual and implementation details are reported in [5].

[3]For more information about this platform please refer to [3], [6] .

978-1-5090-5387-2/16/$31.00 2016 IEEE

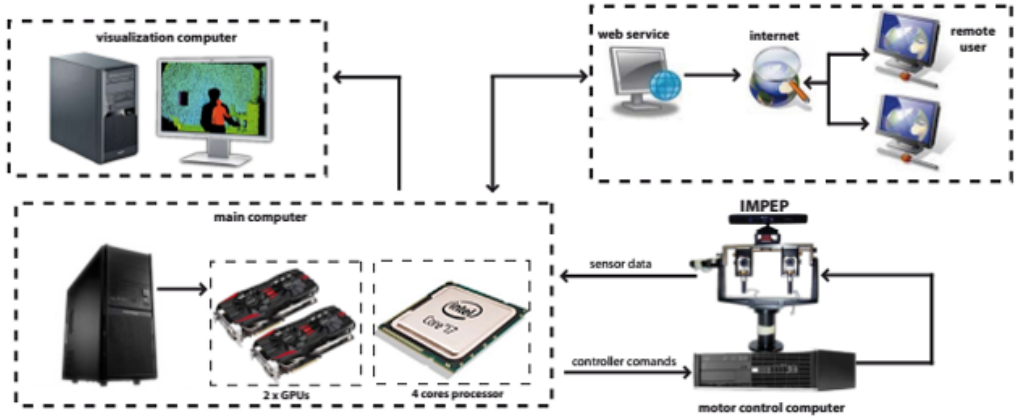

Fig. 2: CASIR-IMPEP system architecture overview [3] only the bottom part of this diagram was originally fully implemented during the duration of the CASIR project, while the top part was developed as an expansion in the scope of the work presented in this text.

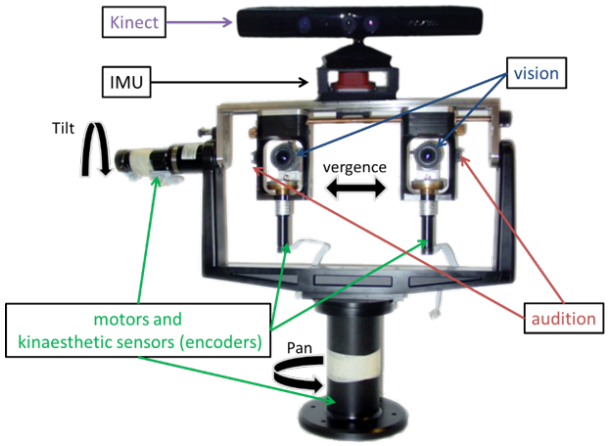

Fig. 3: The Integrated Multimodal Perception Experimental Platform [3], including actuators and respective degrees of freedom, and mounted sensors.

access to remote users through a remote robotic lab.

II. Related Work

As the effort of applying a systematic approach to meeting the demand of implementing features such as those presented in Fig. 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 [7] has a similar architecture to the CASIR framework; however, it deals with a different application scope via a mobile manipulation platform. The iCub simulator was created to complement the iCub project. It is a very specific simulator with an unique architecture, it uses YARP (Yet Another Robot Platform [8]) instead of ROS and a network wrapper for remote access. Another project, "The Construct Sim" [9], consists of a cloud based tool for remote robotic simulation. It 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.

The PR2 and Care-O-Bot were found to possess all of the desired features displayed on Fig 1, 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 does not make hardware available. Finally, for Care-O-Bot the price of every module is provided by the company on request.

The contributions of this work, represented in Fig. 4, resulting of the implementation of an integrated framework boasting the features presented in Fig. 1, consist of providing the full feature set with the widest availability possible. 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. Unlike related work, the framework described in this paper will be developed so as to provide **all the features** of Fig. 1 as **freely available**, and, in the case of the remote lab access by a user external to the local research team, with reservation of timeslots, all the time **ensuring system and hardware security**.

III. Implementation

A. ROS framework for the CASIR-IMPEP platform

ROS is a flexible framework for writing modular robot software, capable of creating complex and robust behaviour in different types of robotic platforms. The ROS framework involves several core concepts, such as *packages*, *nodes*, *topics*, *services* and *messages* – please see [10] and [11] for more information. ROS is both modular and 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.

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)

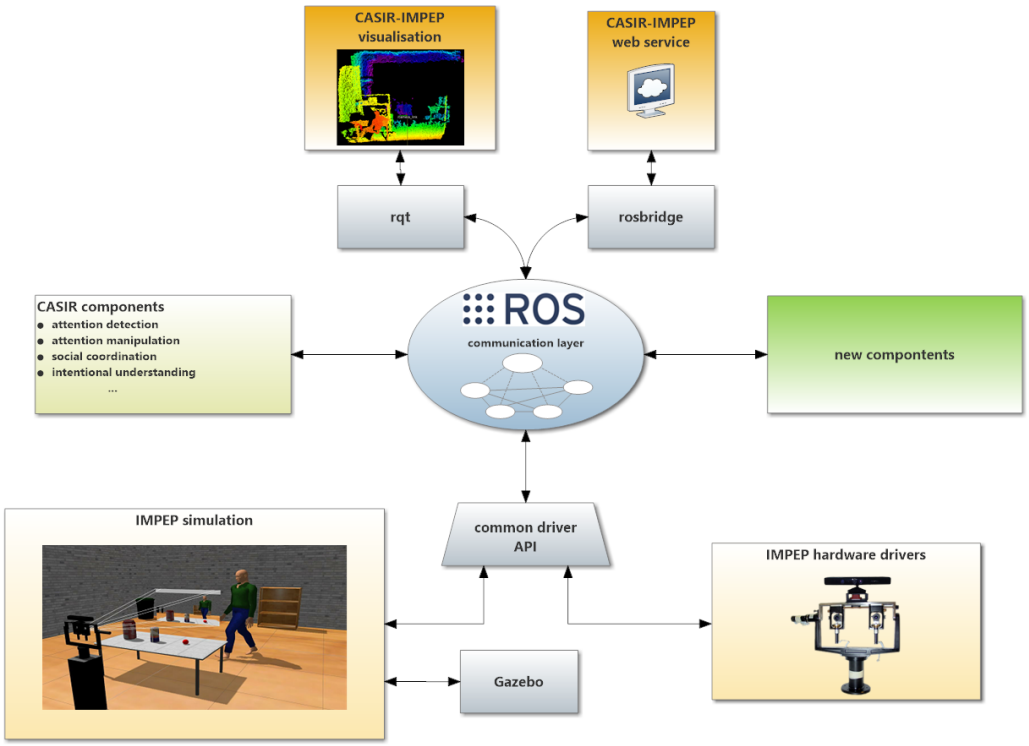

Fig. 4: Conceptual diagram for the IMPEP ROS framework for remote 3D visualisation, control and simulation. The modules in orange refer to the contributions of the work presented herewith, namely the simulator represented by the `impep_simulation`, the hardware access that is not only the IMPEP but also 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.

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, thereby narrowing the universe of development frameworks of interest to Gazebo [13], MORSE [14], V-Rep [15] and Webots [16]. Comparing these frameworks in terms of features, Gazebo and Webots stand out among the group; however Gazebo is more interesting in terms of support infrastructure. 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], this means that 52.9% of functions (or subroutines) in the program were called in tests, and 44.5of branches were executed.

To build the models, several 3D modelling tools were compared, namely Maya, 3ds Max and Blender. These solutions are very similar in features; however, due to the simplicity of the modelling demands of the work reported in this paper, and without the use of complex animations, Blender was deemed to be the most suitable solution.

Applying HMI to robotics is as important as the system itself – it is critical that the user possesses and is familiar with the right tools to work with the system. In order to organise all of this information and give the desired control to the user, the graphic user interface must be designed in order to be simple and intuitive. 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 [10]. 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).

Remote experimental labs 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 [17]. The arrival of new web technologies such as HTML5 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 [18]. Besides displaying ROS information in the form of images, we also need to transmit them over *rosbridge* – to this end, the ROS package named *web_video_server* was 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 [19].

The expected outcome of this work was a unified ROS-supported framework designed so as to attain the objectives laid down in section I, allowing the CASIR attention middleware described in [5] 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.

In this system, 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 Fig 4). The published topics can be subscribed by the attention middleware nodes or seen directly by the remote and local users through the respective GUIs. 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 III-D).

B. Implementation details for the Gazebo-based simulation package

Three packages were developed in order to build a complete robot model that is fully compatible with ROS: `impep_gazebo`, `impep_controller` and `impep_description` – see Fig. 5. The main package, `impep_gazebo`, includes the world file, the avatar scripts 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) which will be the links of our robot. 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. 6. Finally, `impep_controller` includes the actuator models, parameters and publishers.

1) IMPEP simulation – sensors: The IMPEP has three visual sensors: two RGB cameras, and a Microsoft Kinect sensor. 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 [20]. These were modelled as faithfully as possible in the URDF IMPEP model, including their physical characteristics (e.g. mass and body dimensions, the latter also needing to match with the corresponding Blender model characteristics) and technical specifications (e.g. 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

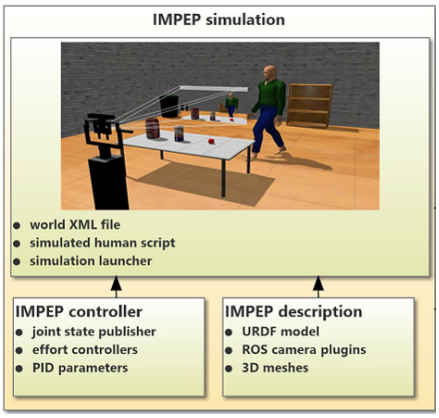

Fig. 5: IMPEP model packages for simulation.

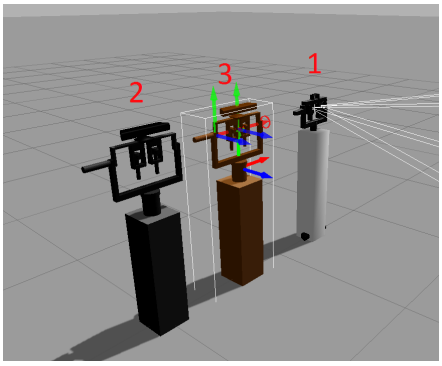

Fig. 6: IMPEP virtual model evolution. Model (1) was the pre-existing, preliminary IMPEP model. Model (2) is the upgraded physical model of IMPEP, completely to scale in terms of mass and dimensions. Finally, (3) represents the final model, with the collision mesh and joint referentials.

Gazebo-ROS plugin named `libgazebo_ros_camera.so` attached to both right and left camera lens models. This plugin is responsible for the publication of camera data to a *rostopic* specified in its parameter definition. Additionally, the effect of Gaussian noise was modelled in order to simulate residual imperfections intrinsic to every real camera.

The depth camera, the Microsoft Kinect V1 RGB-D sensor, already possesses a Microsoft Kinect 3D model natively available in Gazebo that follows the body dimensions of a real Kinect; however, the remainder of the parameters had to be inserted into the model by hand. 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.

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.

2) IMPEP simulation – actuators: The IMPEP includes 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 [21]).

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. The implementation of end of movement sensors consists in creating an upper and lower movement limit in the revolute joints, therefore emulating the function of the kinaesthetic sensors of the real IMPEP. 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 numerous important tasks, namely implementing PID parameters, dealing with publishing joint states, and converting them to TF transforms for *rviz* and other ROS tools.

3) Environmental simulation: In Fig. 7 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 [22].

4) Avatar and interaction simulation: 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.

C. Implementation details for the rqt-based user interface

In most ROS frameworks, spatial visualisation is implemented using *Rviz*. However, in spite of being a very complete tool, using it standalone is not as simple or interactive as required for our system. In order to capitalise on the advantages of *Rviz* while adding increased flexibility in GUI design, *rqt_rviz* was used [10]. 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.

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.

A GUI layout implementing these features is presented in Fig. 8. The plugin used for 2D visualisation is called *rqt_image_view* [10] – it is an rqt version of ROS's *image_view* [10], 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). A third feature of the GUI is the ability to embed a terminal in an interface frame, via the Python GUI plugin *rqt_shell*, which supports a fully functional embedded XTerm [10]. An improvement to the terminal plugin was made, allowing it to display two windows in the same space (with the use of tabs). Finally, we implemented a user-friendly package launcher using an experimental plugin named *rqt_launch*, allowing the user, among other things, to run and stop selected launch files (and individual nodes from the active launch file) chosen via a dropdown menu.

Using the configuration file `.perspective`, 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.

A running instantiation of the GUI is presented in Fig. 9.

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

The web service supporting the CASIR-IMPEP remote lab uses a client-server architecture implemented with *Rosbridge*. 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 [10].

The first implementation step was to set up the server side. 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

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

Fig. 7: Comparison between simulated and real set-up for HRI.

Fig. 8: Mockup of the rqt-based interface. Each frame in the interface 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 in tabs; (5) contains the rqt launcher with the possibility of node selection. Note that all of these frames are reconfigurable on the fly.

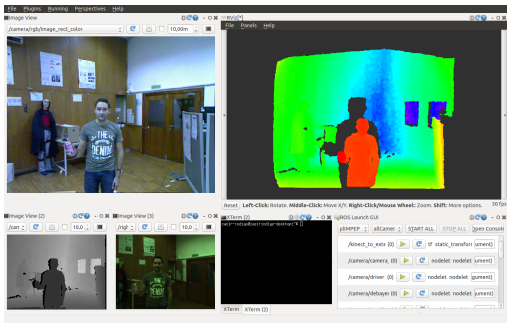

Fig. 9: Instantiation of the rqt-based interface applied to the real system in operational conditions.

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. The user interface for the client side of the web service, on the other hand, was created using a simple HTML file – see Fig. 10. In a lower layer, however, it has JavaScript modules that communicate with Rosbridge through websockets. To create these scripts, we used as a template the work by Blaha et al., 2013 [23], updated them with contemporary plugins and custom parameters so as to conform with our requirements.

An overview of the complete system, from server to client

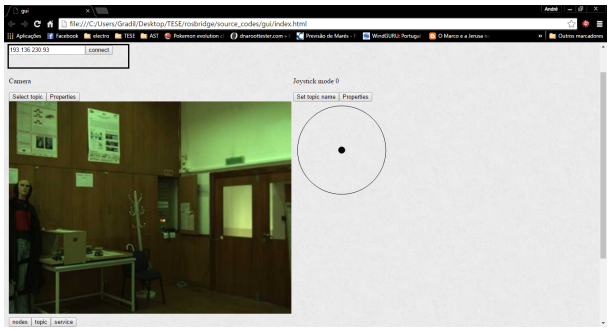

Fig. 10: CASIR-IMPEP remote lab, the HTML web page already connected to the server and streaming one topic.

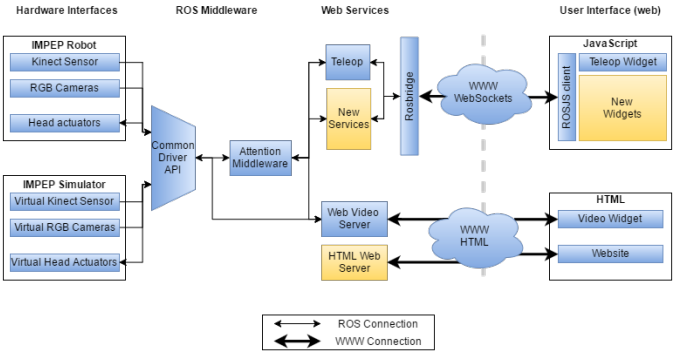

Fig. 11: Complete dataflow diagram of the web service. The modules in yellow represent potential for future expansions.

passing through the communication protocols, can be seen in Fig. 11, including possibilities of expansion. 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. In summary, the interface 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.

IV. Results and Discussion

The developed framework was tested in order to evaluate its performance in full-blown operation(videos including examples of several operational conditions of the system can be found online in https://www.youtube.com/playlist?list=PLfGLccmDrgxsfCZcNaxXS7ztNgnMlbBKd). The core

TABLE I: 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

computational hardware supporting the framework (Fig. 2) is composed by the **main computer**, which includes a 4 core Intel Core i7-3770 @ 3.40GHz CPU, 2 Asus GTX780 3GB DirectCU II OC GPU units, an Asrock Z77 OC Formula motherboard with 16GB DDR3-1600Mhz RAM, a 128GB KINGSTON SSDNow V200, and a Seagate Barracuda 1TB HDD, and also the **visualisation computer**, which is expected not to be as powerful as the main unit. To validate this assumption we used a virtual machine (using VMWare software) emulating a 2 core Intel Core i5-6300HQ CPU @ 2.30GHz, a Gallium 0.4 on SVGA3D GPU, supported by 3GB DDR3-2300MHz RAM and a 76.0 GB hard disk. As for the remote CASIR-IMPEP lab clients, several different machines with various operating systems were used for testing purposes.

1) Simulation and visualisation proof-of-concept: CPU and RAM usage was measured using *htop*[4], while GPU usage was measured using *nvidia-smi*[5].

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. Running experiments in the simulated world were monitored by the user through the IMPEP's cameras and a global simulated external camera view, giving a third-person perspective. As can be seen in Table I, the absence of a Gazebo UI reduced almost 12% of the CPU load despite the need for an extra camera with higher resolution. Effects in memory usage were found to be negligible, while GPU usage, on the other hand, dropped slightly, as expected, since there was no need to render the entire world in real-time.

Exhaustive tests were also conducted to evaluate visualisation performance, 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. 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,

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

[5]nvidia-smi is a command line utility intended to aid in the management and monitoring of NVIDIA GPU devices.

TABLE II: Local vs remote visualisation benchmark comparison, the specifications of both machines are those in section III-D

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

displaying the same topics. Table II shows the results for the two operational scenarios. We found 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 conducted tests for different combinations of these conditions, where we measured the computational weight of both simulation and visualisation running simultaneously, comparing the effects of the lack of UI. Performance was found to be coherent with 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, brought forth important conclusions about the limitations of our system. We found that the major system load comes from the middleware itself. Since the CPU is already overloaded, 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.

2) Remote simulation and control proof-of-concept: In order to benchmark network resource usage, the remote lab was tested through three separate internet connections, specified in Table III. 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; in point 3-Latency of [10].)

TABLE III: 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

Our system was found to be stable for all connections, 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.

V. Conclusion and Future Work

Throughout this text, the overall objective and goals defined in section I were shown to be satisfactorily attained, and, as demonstrated in section IV-1, performance requirements were also satisfactorily met. Every component in the proposed framework acts in a synergistic fashion so as to complement the core nodes in a simple, modular, and optimised fashion, trying in full to minimise resource usage and computational burden of the main computer. A detailed description of the work presented herewith can be found in [24].

Concerning future work, the Gazebo simulator will be improved in terms of simulated human actors, namely by adding improved, more realistic models, for example by resorting to 3D scanning and creating the skeleton and joints accordingly. A suite of action scripts for the animations will also be developed, and later on the development of a user-friendly toolkit to assist with script generation. In terms of hardware inclusion, 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. As for the *rqt* visualiser, the launcher section of the UI will be redesigned in order to make it even more user-friendly, especially on what concerns non-developers, as already shown in Fig. 8. Other improvements will be to develop visualisation screens for the diverse data displays needed for attention middleware development and monitoring (for example, different types of 3D reconstructions, signal waveforms, etc.). Finally, for the remote lab the most important follow-up work will be hosting the web site in a server and to provide a secure, PHP/SQL handshake procedure for access authorisation (i.e. a welcoming homepage, time slot requests to the webmasters, etc.).

References

[1] 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.

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

[3] P. Lanillos, J. N. Oliveira, and J. F. Ferreira, "Experimental Setup and Configuration for Joint Attention in CASIR," Mobile Robotics Lab Institute of Systems and Robotics, Coimbra, Portugal, Tech. Rep. MRL-CASIR-2013-11-TR001, November 2013. [Online]. Available: http://mrl.isr.uc.pt/projects/casir/index.php?menu=5&language=eng&tabela=geral

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

[5] 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)*. IEEE, 2015, pp. 4171 – 4178.

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

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

[8] G. Metta, P. Fitzpatrick, and L. Natale, "YARP: Yet Another Robot Platform," *International Journal on Advanced Robotics Systems*, p. 3(1):4348, 2006.

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

[10] OSRF, "Ros Wiki," [Online; accessed 09-August-2016, last edit 15-September-2016 09:17:13]. [Online]. Available: http://wiki.ros.org

[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, no. 3.2. Kobe, Japan, 2009, p. 5.

[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*. IEEE, 2014, pp. 842 – 849.

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

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

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

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

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

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

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

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

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

[22] 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.

[23] M. Blaha, M. Krec, P. Marek, T. Nouza, and T. Lejsek, "Rosbridge web interface," Department of Cybernetics Faculty of Electrical Engineering, Czech Technical University Technick, 166 27 Prague 6, Czech Republic, Tech. Rep., May 2013.

[24] Gradil, A. , "Development of a Remote 3D Visualisation, Control and Simulation Framework for a Robotic Head," [Online; accessed 07-November-2016]. [Online]. Available: http://ap.isr.uc.pt/archive/651.pdf