Intelligent Networked
robot Systems for
symbiotic Interaction
with children with
impaired
DEvelopment

Architecture Technical Report

André Mateus, Carla Guerra, and Rubén Solera

February 15, 2017

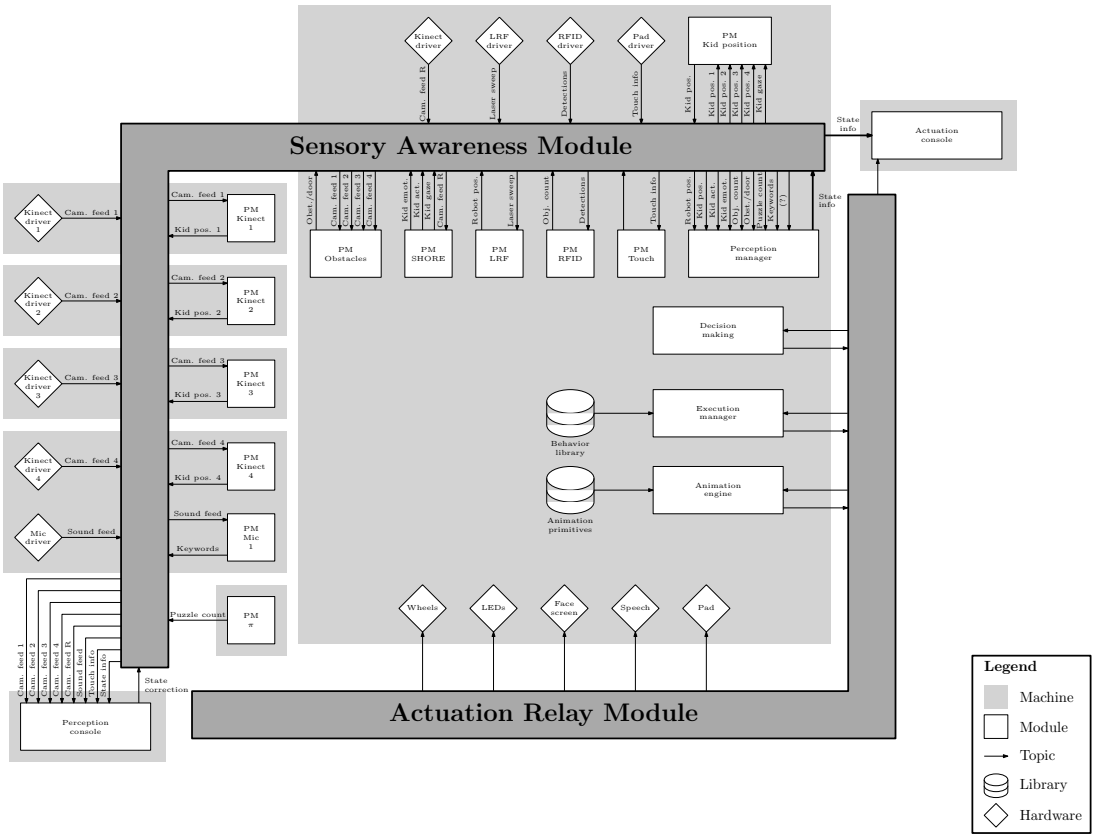

Figure 1: INSIDE Architecture Overview

1 Introduction

INSIDE is a project funded by the Portuguese Fundação para a Ciência e a Tecnologia under the CMU - Portugal Program, and the Information and Communications Technologies Institute (ICTI). As far as research is concerned, INSIDE focus on symbiotic human-robot interactions in joint cooperative activities. In particular, these activities consist on having the robot help the human in tasks the human cannot perform by himself (e.g. complete a puzzle with missing pieces) and the reverse, i.e, the human helping the robot (e.g. open a door). The hardware and software developed in our research will be employed in therapeutic sessions with children with Autism spectrum disorder (ASD).

This report presents the technical aspects of the architecture for the INSIDE Project. A diagram of the complete framework is presented in Figure 1. All software is available at IRSg's gitlab server, on a repository [1] created for the project. The code is presented in the *catkin_ws* folder, the *master* contains the more recent stable version of the code.

The remainder of the report is organized as follows. Section 2 presents the hardware components of the architecture, Section 3 presents a description of the perception modules. Then in Section 4 the actuation modules available are presented followed by the supervisor consoles in Section 5. Information about the ROS messages, services, and actionlibs is presented in Appendixes A, B, and C respectively.

[1] http://dante.isr.tecnico.ulisboa.pt/amateus/INSIDE/tree/master. Login and access to repository are necessary. If you do not have access, please contact the project PI.

Figure 2: INSIDE robot.

2 Networked devices

This section present all the networked devices present in INSIDE's architecture. It starts by giving an overview of the robotic platform used in the last pilots of the project, then list the computers used with some emphasis on their specifications. Finally the remaining hardware (sensors) are described.

2.1 Robotic Platform

The robot used in INSIDE was developed by IdMind, it is a omnidirectional platform with a width of 529mm, a lenght of 684mm, 1058mm of height, and weights 40Kg. It is equipped with a computer onboard, with an i7-6700T processor, and 8GB of RAM. Besides the computer it has the following sensors:

- RFID reader;
- 5m Hokuyo LRF;
- 20m URG LRF (add specific models);
- Orbbec Astra S RGB-D camera;
- HD webcam (add specific model);
- SuperBeam^{TM} Stereo Array Microphone.

It is also equipped with the following actuators:

- 4 Mecanum wheels;
- 10-inch touchscreen;
- RGB LEDs;
- Speakers;

- 10-inch LCD;
- Head Pan (servo).

A depiction of robot from a front and back view is shown in Figure 2.

2.2 Computers

Besides the computer onboard the robot, described above, the architecture consists of six more computers:

- *4 Intel NUC5i7RYH*. All of them are equipped with an i7-5557U processor, 8GB of RAM, and a 120GB SSD Kingston HyperX Savage. These computers are connected to each MS Kinect for XBOX One, and run the software which takes people skeleton tracking information and communicates that data to the robot via wireless (or Ethernet).
- *Dell Inspiron 23*. A 23-inch touch-screen all-in-one computer model 2350 - 3.2GHz Intel Core i5-4210M Processor, 8GB Memory, 1TB Hard Drive. The touch-screen is used to show the perception console.

2.3 Other Hardware

Besides the sensors and actuators onboard the robotic platform, there are several offbaord sensors, which are:

- *4 Microsoft Kinect for XBOX One*. A RGB-D camera based on time-of-flight (ToF) technology. They are used to track the skeleton of people in the environment, based on the proprietary software of Microsoft.
- *Shure BLX Wireless Microphone System*. The wireless microphone system, shown in Figure 3, consists of an earset microphone (Shure MXC153), a UHF wireless bodypack transmitter (Shure BLX1) and a dual UHF receiver module (Shure BLX88). The use of a close-microphone (instead of an ambient microphone or the microphones on the robot) facilitates the recognition of the keywords since its proximity to the therapist makes it less sensitive to other speech or noise sources (kid, robot) and room distortions.
- *Micro Electrical Mechanical System (MEMS) microphone array*. An (approximately) omnidirectional circular array of 8 MEMS microphones placed on the robot's head is used to acquire and record the kids' speech during the therapy sessions. The microphone array, whose main elements are shown in Figure 4, consists of a STMicroelectronics STM32NucleoF446 motherboard, an expansion board and 8 omnidirectional digital MEMS coupon microphones that can be directly plugged to the board with long, flexible wires, giving the possibility to adapt the geometry of the array to the surface of the robot shell. A mini-USB to USB cable is used to connect the STM32NucleoF446 board to the robot's main board, where subsequent processing of the recorded

Figure 3: Shure BLX Wireless Microphone System used for keyword spotting of the therapist speech.

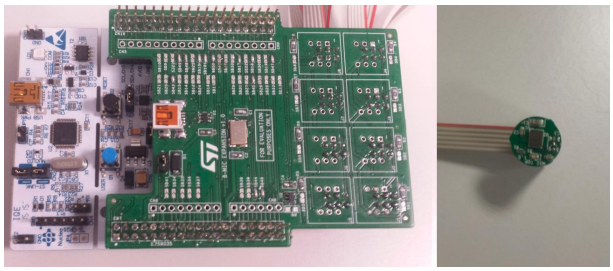

Figure 4: Micro Electrical Mechanical System (MEMS) microphone array. On the left the STMicroelectronics STM32NucleoF446 motherboard, and the expansion board are shown. On the right one microphone out of the 8 is shown.

speech can be performed. The MEMS microphone array provides 8 main audio channels, each of them associated to a specific direction with respect to the array/robot, plus an additional channel with the result of an on-board processing of the 8 main audio channels (configurable via firmware).

- *Samsung LD220Z Lapfit Touch*. A 22-inch Full HD touch-screen used to show the actuation console.

3 Perception Modules

This section presents details on all perception modules used in the architecture. All modules are presented in the ROS package *inside_perception*, available in the ROS catkin workspace structure. In order to execute the modules described bellow execute the command

roslaunch inside_launch main_to.launch,

in the robot. For more details about the code tree refer to the git repository.

3.1 Child Position

Child position is obtained from the Kinects' for XBOX One skeleton information. This version of the Kinect allows tracking the skeleton (body) of up to six people simultaneously. As in the older version, the computation of body positions is performed onboard the sensor and both the algorithm and the

dedicated board which runs it are patent protected. Nevertheless, the information is easily accessible through the Kinect for Windows SDK 2.0[2]. However, the SDK is only available for the Microsoft Windows operating system. With that in mind the child position module was implemented as a Windows program, which connects to a Kinect v2, retrieves the skeleton information, and the floor plane with respect to Kinect's reference frame. The interface with ROS is performed resorting to YARP ports[3]. This software is seen in the ROS architecture as the node

- /kinect#/people_tracker – This node corresponds to boxes *PM Kinect #* in Figure 1 (# represents the number of the Kinect, 1 to 4).

  Published Topics

  - /kinect#/tracked_people - Contains the body information of all people tracked by the Kinect, and the floor plane. Type yarp/KinectTrackedPeopleMessage.

3.2 Child Velocity

This module is under develpment (is missing in Figure 1).

3.3 Child Orientation wrt robot

TBD – is not decided if it will be present in the final architecture.

3.4 Robot Pose

The robot's pose (*PM LRF* in Figure 1) in 2D is estimated resorting to the Adaptive (or KDL-sampling) Monte Carlo Localization proposed by Dieter Fox (insert citation). This is implemented as the *amcl*[4] ROS package. This package performs robot pose estimation in 2D resorting to a laser-based map, laser readings, and the robot's transformation tree. The outputs consist of a topic containing the robot pose (*amcl_pose*) with respect to the map reference frame, the particle cloud, and the corrected transformation between the *odom* and *map* reference frames. For further details on the package see the official documentation.

3.4.1 Discrete Robot Position

Both the perception and actuation consoles display the robot position in discrete regions of the map. In order to convert the absolute robot pose into those discrete regions a ROS node was implemented, which defines the regions and check in which the robot position is contained. Node:

- robot_position.py

  Parameters

[2]https://msdn.microsoft.com/en-us/library/dn782041.aspx
[3]http://www.yarp.it/yarp_with_ros.html
[4]http://wiki.ros.org/amcl

– zones - yaml file containing a python dictionary of the zone definition, structured as follows: {'zone': [coordinates of the four corners limiting the region]}.

Subscribed Topics

– amcl_pose - contains absolute pose of the robot in the map reference frame. Type: geometry_msgs/PoseWithCovarianceStamped[5].

Published Topics

– robot_position - string with a discrete zone where the robot is. Type std_msgs/String[6].

3.5 Obstacles

Obstacle (PM Obstacles in Fig. 1) obstruction detection is performed based on the front laser range finder readings. This consists on counting the number of points in a user defined box (defined immediately in front of the robot). Obstruction is considered to occur, when the number of points inside the box is higher than a certain threshold.

The software is implemented as two ROS python nodes, located on the package inside_perception, folder scripts. These nodes are:

- obst_detector.py - performs the procedure described above.

  Subscribed Topics

  – scan - Laser readings. Type: sensor_msgs/LaserScan[7].

  Published Topics

  – navigator/obst_count - Number of points inside the box. Type: std_msgs/UInt32[8];

  – navigator/obst_scan - Points inside the box in laser reading format. Type: sensor_msgs/LaserScan.

- obstacles_detection-py - receives the number of points in front of the robot, performs the thresholding and publishes the result as a Boolean for the perception manager.

  Subscribed Topics

  – navigator/obst_count. Type: std_msgs/UInt32;

  Published Topics

  – door - Boolean representing the door state, opened (false) and closed (true). Type: std_msgs/Bool[9];

  – obstacle - Boolean representing the obstacle state blocking (true) free (false). Type: std_msgs/Bool.

  Service Servers

[5] http://docs.ros.org/kinetic/api/geometry_msgs/html/msg/PoseWithCovariance.html
[6] http://docs.ros.org/api/std_msgs/html/msg/String.html
[7] http://docs.ros.org/api/sensor_msgs/html/msg/LaserScan.html
[8] http://docs.ros.org/jade/api/std_msgs/html/msg/UInt32.html
[9] http://docs.ros.org/api/std_msgs/html/msg/Bool.html

– enableobstacle - This service is used to activate/deactivate the obstacle detection, thus preventing continuously running it. Type - inside_msgs/EnableObstacle.

3.5.1 Door

3.6 Keyword Spotting

The keyword spotting system (KWS) based on the AUDIMUS ASR engine [1, 2] has been used to recognize certain informative (task related) keywords. This module is represented by *PM Mic 1* in Figure 1.

AUDIMUS is a hybrid HMM/MLP speech recognition system developed by L2F/INESC-ID in C/C++. It combines the temporal modelling capabilities of Hidden Markov Models with the pattern discriminative classification capabilities of Multi-Layer Perceptrons. The acoustic model combines the phone posterior probabilities generated by three phonetic classification branches (MLPs) trained on 26 Perceptual Linear Prediction (PLP), 26 Log-RASTA, and 28 Modulation Spectrogram (MSG) acoustic coefficients, respectively. An acoustic temporal context of 13 (PLP, Log-RASTA) or 15 (MSG) frames is used - MLP size. The decoder module is based on the Weighted Finite-State Transducer (WFST) approach. The configuration parameters of the AUDIMUS-based KWS system are defined in file asr_pt_fc_keyspot.xml.

The AUDIMUS ASR engine may either use general (out-of-domain) or specific (in-domain) language models. In this case, a specific equally-like 1-gram language model formed by all the possible target keywords and a competing speech background/filler model is used [3]. Currently, the keywords have been defined in file pt.grxml.

The AUDIMUS dll (Windows) is invoked through a Java interface; as a result, the KWS system is continuously running in one of the Intel NUC PC's. It gets the speech signal from the audio input (1 channel, 16000 samples/sec, 16 bits/sample), converts it to an appropriate internal format and then performs the keyword spotting process. The keywords are presented in the standard output as they are recognized by the system. The communication with the ROS environment is performed resorting to Java ROS bridge [10], which uses Jetty 9.

- diststart7On.bat - bat file to run the Audimus System on the intelNUC #4.

  Published Topics

  – keywords - Recognized keyword by the AUDIMUS system. Type: std_msgs/String;

  Parameters

  – backgroundPenalty - Weight of the background model with respect to the keyword models in the decoding stage. This parameter can be used to make the system more prone towards keyword detections or rejections [3].

3.7 Number of objects

Performed resorting to the RFID reader on the robot (not implemented yet).

[10] https://github.com/h2r/java_rosbridge

3.8 Child Activity

Gesture recognition, particularly for the *Movie* game.

3.9 Child Emotions

It will be performed resorting to Fraunhofer's Shore^{TM} [11] (under development).

3.10 Perception Manager

As the name indicates this module is responsible for managing all perceptions. Every time a new event (change in the state of any perception variable) is received (via ROS topics), it is send to the Perception Console. Where the operator may correct or confirm the data, by means of ROS services. After receiving the validated data, the Manager assesses if the value of the variable was indeed changed and if so it is sent to both the decision making module, and the Actuation Console (via a ROS topic.) It is implemented in Python as a ROS node:

- perception_manager.py

  Subscribed Topics

  - door - Boolean representing the door state, opened (false) and closed (true). Type: std_msgs/Bool;
  - obstacle - Boolean representing the obstacle state blocking (true) free (false). Type: std_msgs/Bool.
  - keywords - Recognized keyword by the AUDIMUS system. Type: std_msgs/String;
  - robot_position - string with discrete robot position. Type std_msgs/String.

  Published Topics

  - console/door - Info to perception console. Type: inside_msgs/Door.

  Services Servers

  - correct/door - service server for handling correction of door state. Type: inside_msgs/CorrectDoor.

4 Actuation Modules

This section presents the actuation modules (functionalities) that the robot can perform. This comprehend robot navigation, animation, and speech (text-to-speech). Consider henceforth that a functionality consist on the following modules, and behaviors correspond to a higher level function, that may be a set of basic functionalities.

[11] https://www.iis.fraunhofer.de/en/ff/bsy/tech/bildanalyse/shore-gesichtsdetektion.html

4.1 Robot Navigation

As far as robot navigation is concerned two modules are available. Those are the ROS 2D navigation stack[12] and professor's Rodrigo Ventura navigation stack[13]. Both approaches consist of the set of ROS packages which take information from odometry, sensors and a navigation goal pose, and output velocity commands to drive the robot to the desired location. The former is native to the ROS middle ware, meaning all the packages in this stack are available from ROS repositories. Among those packages there is *move_base*[14], which provides the ROS interface for configuring, running and interacting with the remaining components of the navigation stack. When *move_base* receives a new goal pose, it calls the global/path planner (e.g. A^*), which computes a path. Then the path is sent to the local planner, i.e, the module responsible for computing the velocity commands, (e.g. Dynamic Window Approach), which will compute the velocity commands to send to the robot wheels controllers. Both the path planning step and the local planner take into account two costmaps[15] (a global, and a local map), that are updated with the information from the sensors.

The latter navigation stack provides an API to provide autonomous navigation functionality to both differential and omnidirectional robots. It assumes there is a static map published by the *map_server* node, and the robot is localized by means of publishing in the *tf* system. The interface provides a set of *actionlib*[16] servers which allow goal pose specification as:

- fixed goal pose;
- fixed goal pose given by a symbol;
- dynamic goal pose, published as a frame in *tf*[17];
- velocity reference to be tracked;
- body heading control published in a topic.

Motion Planning is performed with Fast Marching Method and guidance with Dynamic Window Approach. For a detailed description of the system check the web page.

4.2 Robot Animation

Under development

4.3 Speech

Regarding the speech production module, our goal there is to endow the robot with a natural and engaging vocal interface to interact with the children. For that purpose, two different approaches have been adopted and implemented:

[12] http://wiki.ros.org/navigation
[13] http://dante.isr.tecnico.ulisboa.pt/yoda/isr_cobot/tree/master/rosbuild_ws
[14] http://wiki.ros.org/move_base
[15] http://wiki.ros.org/costmap_2d
[16] http://wiki.ros.org/actionlib
[17] http://wiki.ros.org/tf

- DIXI Text-to-Speech Engine - DIXI [4] is a concatenative-based text-to-speech synthesizer based on the well-known Festival Speech Synthesis Systems. DIXI has been jointly developed by L2F/INESC-ID and VoiceInteraction S.A. in C/C++, and includes several modules performing text splitting and normalization, part-of-speech tagging, prosodic phrasing, grapheme to phone conversion, post-lexical analysis and waveform generation by means of a unit selection and concatenation approach. The synthesizer receives a raw text string and returns a file with the corresponding synthesized speech waveform (22050 samples/sec., 16 bits/sample, 1 channel, wav format) and the data required for lip synchronization (a sequence of phonemes with their corresponding temporal durations). For convenience, the TTS service finally converts the wav file to an mp3 format before playing it on the robot's speakers.

- Pre-recorded speech utterances. Due to the difficulty of endowing synthesized speech with natural emotions, which are very important for an improved engagement experience of the children with the robot, a set of pre-recorded speech utterances (44100 samples/sec., 16 bits/sample, 1 channel, wav format) is used for the robot's vocal output. In this case, the robot just plays the speech file corresponding to the selected utterance. To generate the data required for lip synchronization, the AUDIMUS ASR system [1] using monophone acoustic models is employed in this case to perform a forced alignment (segmentation) of the pre-recorded speech files with their corresponding transcriptions.

The speech production module is implemented following a client/server architecture, with the server running as a local service in the robot itself. The server is implemented as a Java executable that invokes the DIXI dynamic-link library (Linux dll). In order to execute the server go to the folder *speechSynthesizer* in the project's git repository and run the following command

./run.

The propeties of the server can be changed by editting the file *tts.properties*, in the *speechSynthesizer/dist2* folder.

The client is implemented as a ROS node in package *inside_speech*. This node makes a POST request to the server, receiving a JSON containing the path to the audio file, the correspondent phonemes, and their temporal duration. After decoding the JSON the audio is played using vlc, with a non-blocking system call. Alongside the audio reproduction the robot executes the lips movement corresponding to each phoneme in the respective time duration, while keeping the emotional display. **Note:** The INSIDE robot cannot perform lip movement yet.

Nodes

- speech_l2f_local.py - Send Post request to local speech production server and plays the resulting audio file. *Services Servers*

  - speech - Receives a string (sentence to speak) and the emotion (current emotion of the robot) returns a boolean, which is true if the sentence was said. Type - inside_speech/SpeechSrvEmo.

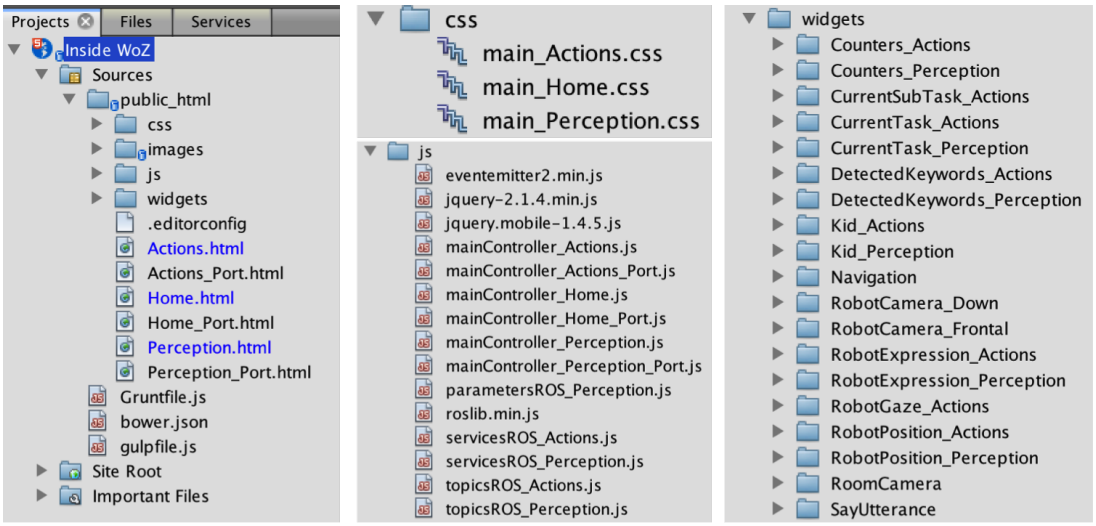

Figure 5: NetBeans Project Folders.

5 Supervisor Consoles

The communication between the whole system and the wizards of oz is made through two consoles - one console regards the perceived information and is controlled by one of the wizards (see Figure 1 - bottom left corner), the other one regards the actuation part (see Figure 1 - upper right corner) and is controlled by another wizard. Only corrective feedback is expected through these consoles and not full control over the entire system by the wizards.

5.1 Consoles Development and Files Organisation

The consoles were developed using NetBeans 8.1 IDE, including Javascript, HTML5 and CSS files. These files are organised as shown in Figure 5. All of them are duplicated, one regarding the perception console, another one regarding the actuation console. There is one folder 'CSS' with the CSS files, which say how the HTML elements are to be displayed on screen - colours, sizes, positions, etc. The 'images' folder saves the images used in the project. The folder 'js' has all the javascript files - the mainControllers, with the definition of general functions, the files where the communication with ROS is made through parameters, topics and service servers and others already implemented downloaded from the internet. The final folder, 'widgets', has several subfolders, each one containing the HTML file regarding a specific module of the console - each of these modules is explained in the following sections. The main files are the Perception.html and Actions.html, which have the overall definition of the perception and actuation console pages. A lot of these files are duplicated with the suffix '_Port' - they represent the same but are translated to Portuguese (most of them are not up-to-date and will probably be deleted since they are not going to be used). A 'Home' page was also developed (and has its html, css and mainController files), but will probably also be deleted due to no use.

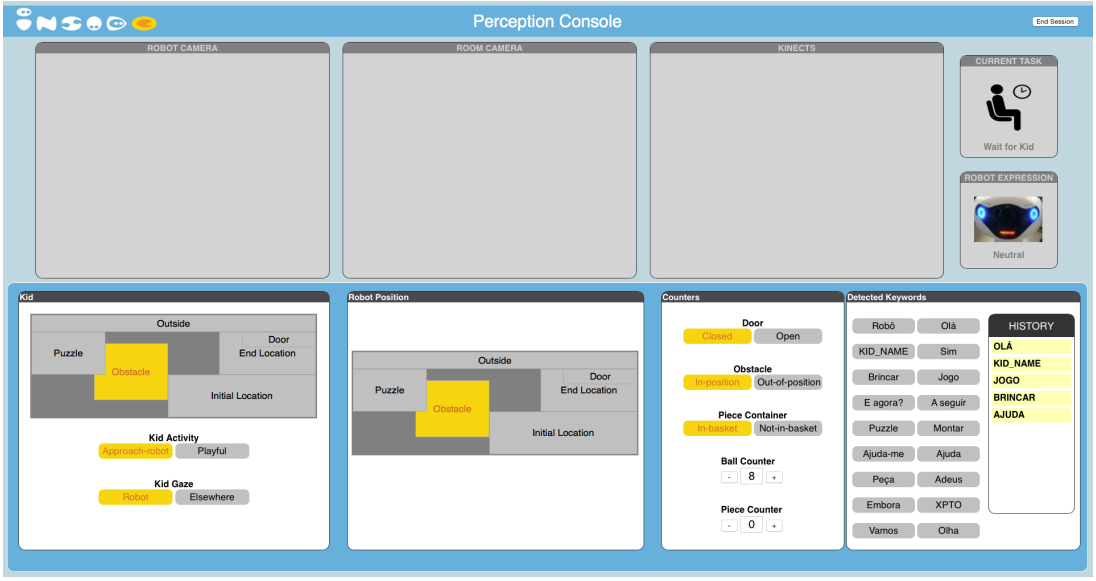

Figure 6: Perception Console.

5.2 Perception Console

The perception console shows the wizard what is happening through the cameras and allows him to correct the processed information that comes from the system - Figure 6. Thus, it includes two parts - an upper part that shows the 'raw data' coming from the robot cameras and another one placed in the room, as well as the task and the robot expression. The bottom part is perception information processed by the system such as positions of the kid and robot, counters and keywords detected. When the system detects any change in the perception modules, the corresponding option starts to blink and changes its color to yellow for 3 seconds, during which the wizard can correct it by clicking on the correct option. If there is no correction, the system assumes its perception is correct. The wizard can change this option anytime he wants by clicking on the new wanted option. All the information presented in the perception console is listed in the next subsections.

5.2.1 Robot Frontal Camera

Shows the view from the robot frontal camera.

Parameters

- frontal_camera - string with URL to the frontal camera of the robot.

5.2.2 Robot Down Camera

Shows the view from the robot down camera, to the basket - to check the balls and the piece of the puzzle.

Parameters

- down_camera - string with URL to the down camera of the robot.

5.2.3 Room Camera

Shows the view from a camera placed in a corner of the room.

Parameters

- roomCamera - string with URL to the camera placed in the room.

5.2.4 Current Task

Shows the current task of the system.

Subscribed Topics

- task - message.data - string with the current task of the system. Type std_msgs/String.

5.2.5 Robot Expression

Shows the current expression the robot is performing.

Subscribed Topics

- robot_expression - message.data - string with the expression the robot is playing. Type std_msgs/String.

5.2.6 Kid Position

Shows the position of the kid in a map with 6 discrete regions: Outside, Initial Location, Mid Room, Obstacle, Puzzle, Door.

Subscribed Topics

- kid_position - message.new - string with the discrete region on the map where the kid is perceived. Type inside_msgs/KidPosition.

Service Servers

- confirm - perception: Kid Position- confirms if the perceived position of the kid is correct. Type inside_msgs/ConfirmPerception.
- kid_position - real_value: corrects the perceived position of the kid. Type inside_msgs/CorrectKidPos.

5.2.7 Kid Gaze

Shows whether the kid is looking at the robot or elsewhere. (to integrate)

5.2.8 Robot Position

Shows the position of the robot in a map with 6 discrete regions: Outside, Initial Location, Mid Room, Obstacle, Puzzle, Door.

Subscribed Topics

- robot_position - message.new - string with the discrete region on the map where the robot is perceived. Type inside_msgs/RobotPosition.

Service Servers

- confirm - perception: Robot Position- confirms if the perceived position of the robot is correct. Type inside_msgs/ConfirmPerception.
- robot_position - real_position: corrects the perceived position of the robot. Type inside_msgs/CorrectRobotPos.

5.2.9 Door

Shows whether the door is open or closed. (to integrate)

5.2.10 Obstacle

Shows whether the obstacle is blocking the robot (in-position) or not (out-of-position).

Subscribed Topics

- obstacle - message.blocking - boolean variable set to true when the obstacle is blocking the robot, set to false when it is not. Type inside_msgs/Obstacle.

Service Servers

- confirm - perception: Obstacle - confirms if the blocking obstacle perception is correct. Type inside_msgs/ConfirmPerception.
- obstacle - blocking: corrects the existence or not of an obstacle (it is a boolean - true when there is an obstacle blocking the robot, false when there is not). Type inside_msgs/CorrectObstacle.

5.2.11 Piece Container

Shows whether the missing puzzle piece is still in the robot's basket or not. (to integrate)

5.2.12 Ball Counter

Shows the number of balls that are in the basket. (to integrate)

5.2.13 Piece Counter

Shows the number of pieces that were already correctly placed in the puzzle.

Subscribed Topics

- piece_counter - message.num_pieces - the number of pieces that were detected as correctly put in the puzzle. Type inside_msgs/PieceCounter.

Service Servers

- confirm - perception: Piece Counter - confirms that the detected number of pieces in the puzzle is correct. Type inside_msgs/ConfirmPerception.

- piece_counter - num_pieces: corrects the number of pieces in the puzzle. Type inside_msgs/CorrectPieceCount.

5.2.14 Detected Keywords

Shows, from a list of keywords, when one of the words was detected. When detected, the word blinks and changes its color to yellow for 3 seconds, during which the wizard can reject that detection double-clicking the word. If there is no correction, the system assumes the detection is correct and adds the word to the history list. When the wizard hears some keyword that was not detected by the system, he can add it to the list by clicking on the correct word.

Subscribed Topics

- keywords - message.keyword - string with the keyword detected. Type inside_msgs/Keywords.

Service Servers

- confirm - perception: Detected Keywords - confirms if the detected keyword is correct. Type inside_msgs/ConfirmPerception.

- keywords - rejected: boolean variable to reject or not the keyword detected by the system; keyword: string variable to add a new keyword the system did not recognised. Type inside_msgs/CorrectKeywords.

5.2.15 End Session

It is a button that finishes the interaction and saves the log files.

5.3 Action Console

The action console shows the actuation wizard the already processed and corrected perception information coming from the perception console (this part the actuation wizard cannot control anymore) - Figure 7. The correction on the actuation console regard the current task and sub-task of the system, the robot position, gaze and expression and, finally, the utterance to perform. Each action performed is added to the history list also shown. Similarly to the perception console, when the system is changing something in the action modules, the corresponding option starts to blink and changes its frame to yellow for 3 seconds, during which the wizard can correct it by (continuous) clicking on the correct option. If there is no correction, the system assumes its decision is correct. The wizard can change this option anytime he wants with a (continuous) click on the new wanted option. A continuous click is required when changing the task, subtask or the robot position, in order to prevent more flagrant behaviours by mistake. All the information presented in the action console is listed in the next subsections.

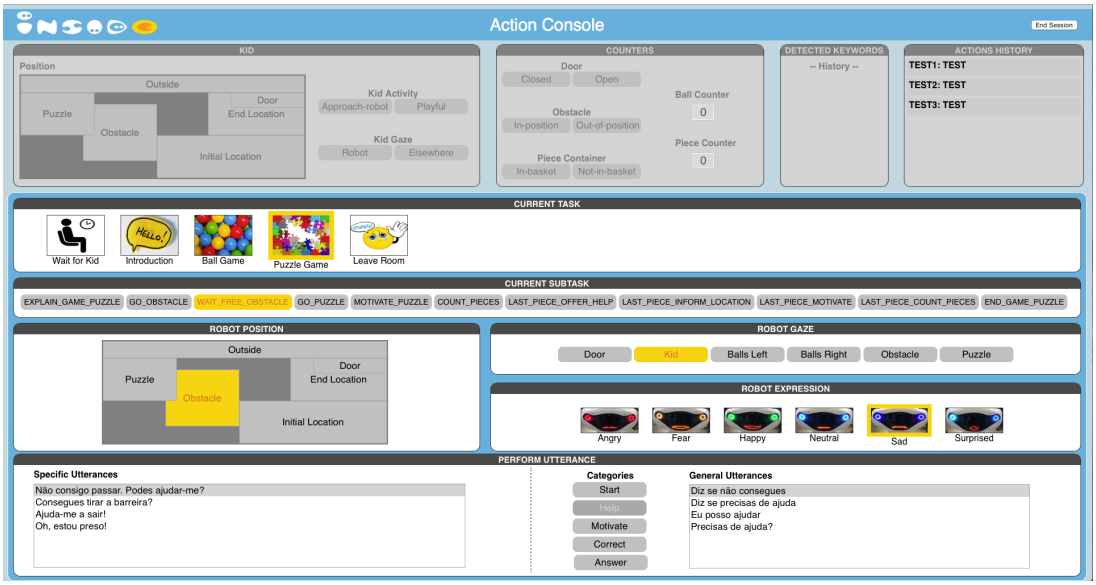

Figure 7: Actuation Console.

5.3.1 Corrected Data from the Perception Console

Shows the already processed and corrected perception information coming from the perception console and cannot be controlled here.

Subscribed Topics

- corrected_info - message.keywords, message.piece_counter, message.obstacle, message.kid_position. Type inside_msgs/StateVar.

5.3.2 Current Task

Shows the task the system is currently on: wait for kid, introduction, ball game, puzzle game or leave room. (to integrate)

5.3.3 Current Subtask

Depending on the current task, this shows the respective list of subtasks and the one the system is currently on. (to integrate)

5.3.4 Robot Position

The perceived location of the robot comes already corrected from the perception console, but the actuation wizard can change the robot's position anytime with a continuous click on the new wanted location.

Subscribed Topics

- corrected_info - message.robot_position - shows the corrected perceived location of the robot in the map. Type inside_msgs/StateVar.

Service Servers

- change_robot_position - corrected_position: changes the position of the robot. Type inside_msgs/CorrectRobotPositi

5.3.5 Robot Gaze

Shows where is the robot looking at: door, kid, balls at the left, balls at the right, obstacle or puzzle. (to integrate)

5.3.6 Robot Expression

Shows the expression the robot is showing: angry, afraid, happy, neutral, sad, surprised. (to integrate)

5.3.7 Perform Utterance

Given the task and subtask selected, this shows the list of possible utterances to perform (loaded from a text file with the complete list) and the one the system will perform next. If the wizard wants some utterance to be performed by the robot, he has only to click on the preferred option. To facilitate the preferred utterance to be found, the list is divided in specific utterances (subtask specific) and general utterances (independent of the task/subtask and divided into 5 categories - start, help, motivate, correct and answer).

Subscribed Topics

- robot_utterance - message.data - shows the utterance the system will perform next. Type std_msgs/String.

Service Servers

- timeout_utterance - timeout - boolean variable to accept the utterance to perform. Type inside_msgs/TimeoutUtteran
- utterance - utterance - selects an utterance to be performed by the robot. Type inside_msgs/CorrectUtterance.

5.3.8 End Session

It is a button that finishes the interaction and saves the log files.

6 Decision Making

Not defined yet!

References

[1] H. Meinedo, D. Caseiro, J. Neto, and I. Trancoso. *AUDIMUS.MEDIA: A Broadcast News Speech Recognition System for the European Portuguese Language*. PROPOR, pp. 9-17. Faro, Portugal, 2003.

[2] H. Meinedo, A. Abad, T. Pellegrini, J. Neto, and I. Trancoso. *The L2F Broadcast News Speech Recognition System*. FALA, pp. 93-96. Vigo, Spain, 2010.

[3] A. Abad, A. Pompili, A. Costa, and I. Trancoso. *Automatic word naming recognition for treatment and assessment of aphasia*. Interspeech. Portland (OR), USA, 2012.

[4] S. Paulo, L. C. Oliveira, C. Mendes, L. Figueira, R. Cassaca, C. Viana, and H. Moniz. *DIXI – A Generic Text-to-Speech System for European Portuguese*. PROPOR, pp. 91-100. 2008.

A Topics Types

- **yarp/FloorPlane.msg**

  float64 x

  float64 y

  float64 z

  float64 w

  //values defining the floor plane equation, $(a, b, c) \in \mathbb{R}^3 : ax + by + cz = w$.

- **yarp/JointROSmsg.msg**

  int64 typeNum

  string typeStr //name of the joint.

  geometry_msgs/Pose pose //pose of the joint w.r.t. is parent joint.

- **yarp/Body**

  string trackID //id of the joint (given by the Kinect).

  JointROSmsg[25] joints //vector containing the joints of a detected person.

- **yarp/KinectTrackedPeopleMessage.msg**

  FloorPlane floorPlane

  Body[] bodies

  //vector of deteced people. std_msgs/Header[18] header

B Service Types

- **inside_msgs/EnableObstacle**

  string obstacle string stating, which object is blocking the robot (door, or piece).

  —

  bool received

- **inside_speech/SpeechSrvEmo**

  string data

  string emotion

  —

  bool success

[18] http://docs.ros.org/api/std_msgs/html/msg/Header.html

C Actionlib

Detail actionlib used