MOnarCH

Multi-Robot Cognitive Systems Operating in Hospitals

Grant FP7-ICT-2011-9-601033

Deliverable D2.1.1 (update)

MOnarCH Robots Hardware

Version: 2

Dissemination level: PU

Date of preparation: December 2014

Editor: Paulo Alvito (IDM)

Contributors: Paulo Alvito, Carlos Marques and Paulo Carriço (IDM)
Marco Barbosa, João Estilita and Daniel Antunes (ST)
David Gonçalves (YDR)

Table of Contents

1. Introduction

This document is an update of deliverable D2.1.1, the description of MonarCH's robots hardware.

Deliverable D2.1.1 described the MOnarCH hardware platform that was created during the first year of the project. The robots were developed and assembled in three phases. The first phase included the development and assembly of the platform base with its mechanics and electronics. The second phase was the development and assembly of a top structure with high-level devices, taking in consideration the design expectations. The third phase was the conclusion of shell design, production and assembly over robot platform. Fig. 1 depicts the different phases of the development of the MOnarCH robots.

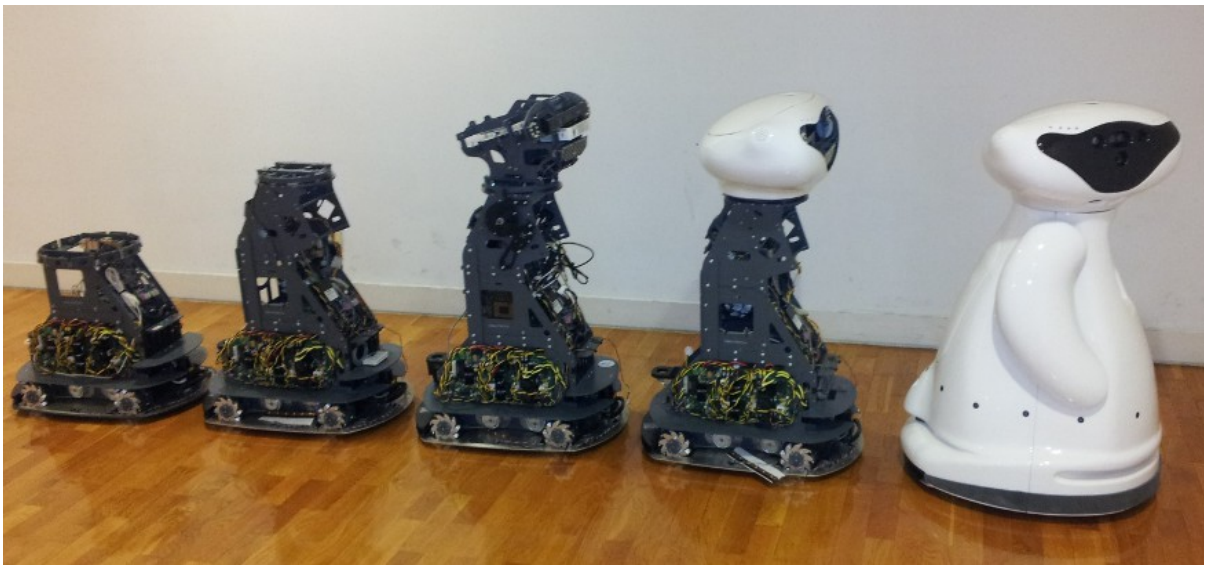

Figure 1: MOnarCH Robot development process.

The MOnarCH project includes two types of robots mounted over a similar platform base: a more sophisticated one targeting social interactions with kids/people (aka SO) and a simpler one that will be used to increase the perception of the environment (aka PO). The two robot types are depicted in Fig. 2.

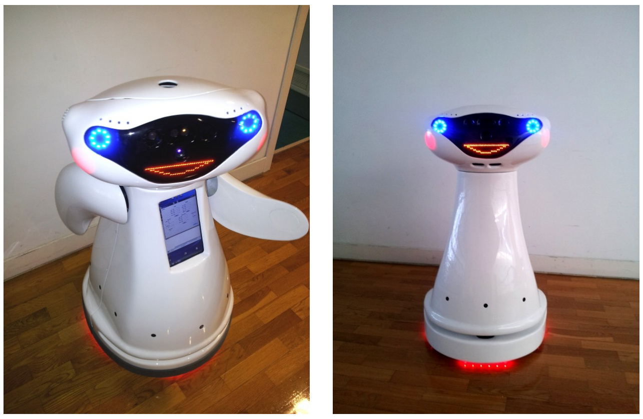

Figure 2: MOnarCH's robots. Left: SO robot. Right: PO robot.

The structure of this document is as follows:

- MOnarCH Robot Features and Components - This chapter describes the main features of the MOnarCH robots. It gives an overview of the different components that have been integrated, their function and placement on the robot;
- MOnarCH Design and Mechanics – This chapter shows the inner MOnarCH robot mechanics and gives an overview of the shell developmental and integration;
- MOnarCH Robot Architecture - The adopted power and communication architecture is explained. The chapter begins with a description of how the robot components are powered. A table with the component energy distribution shows the distribution of the battery power consumption. The last two sub-chapters explain the low-level communication, showing the way the developed electronics communicate with each sensor and actuator, and the high-level communication between the onboard computers, interaction and navigation sensors and actuators;
- MOnarCH Robot Electronics - All of the electronics that have been projected, designed, produced and programmed within the project are explained;
- Charging Docking Station – Shows the developed charging docking station and the way the MOnarCH robots can perform the docking;
- Working with the Robot – Explains how to operate the robot and how to charge the robots batteries;
- Mbot_ROS – How to install and use the Mbot_ROS library, experiments and results;
- Additionally to the manual, three technical annexes have been updated:
  - Board Controllers Software Protocol - This section explains the low-level control protocol implemented between the onboard navigation computer with the Sensor&Management, Motor Board and the IMU and Interaction computer with the Interaction board;
  - Robot Platform CAD Drawings – This section shows the position of each part of the robot with relation to the centre of the robot;
  - Robot CAD Drawings – This section shows the position of each device to the centre of the robot.

2. MOnarCH Robot Features and Components

The MonarCH robots are omnidirectional robots based on four Mecanum wheels actuated by four independent motors. The use of this kind of kinematics substantially increases the manoeuvrability and performance of the platform.

The development and assembly of the MOnarCH robots was executed in two main phases: one first phase which included the platform base mechanics with the motors, batteries and low-level electronics; a second phase which included high-level devices mounted over an upper structure and the shell.

Deliverable D2.1.1 covered the first phase of the development of the robots. This update covers the second phase of the process.

Two types of robots were developed. The PO robots will have as primary goal to act as active sensors and the SO robots will target social interactions. As aforementioned, the SO and PO robots are built over the same platform base, differing in the onboard equipment and external appearance. This update gives a description of all the integrated components that are present in the SO robot, the more sophisticated one. Nevertheless, the difference between the PO and SO robots will be described in the first two sub-chapters.

The structure of this Section is as follows:

- MOnarCH Robots Main Features – Gives a quick look over the SO and PO main features;
- MOnarCH Robot Devices – Gives an overview of the computers present in the SO and PO robots, devices connected to them and the way they connect to each other;
- MOnarcH Sensors - The robot is equipped with perception, navigation, interaction, environment and low-level safety sensors. This section describes the sensors included on the robots, there function and placement;
- MOnarCH Actuators - The robot will be equipped with locomotion and interaction actuators. This section describes the actuators included in the robots, there function and placement.

2.1. MOnarCH Robots Main Features

The SO and PO robots main features are the following:

- Robot kinematics: omnidirectional with 4 Mecanum wheels
- SO Robots weight: 49 Kg
- PO Robots weight: 44 Kg
- Battery autonomy: >3 hours
- Maximum Velocity: 2.5 m/s
- Acceleration: 1 m/s2 (low-level programmed)
- Emergency Stop Acceleration: 3.3 m/s2 (low-level programmed)
- Size (H x W x L): 102 x 57 x 67 cm

2.2. MOnarCH Robot Devices

The SO robots have two onboard computers:

- The Navigation Computer (aka PC-Nav). This computer is running Linux OS (Ubuntu), and is responsible for navigation, localization, system control and actuation of some low-level interaction devices, like LEDs.
- The Human Robot Interaction Computer (aka PC-HRI). This computer is presently running Windows 7 OS, and is responsible for the control of the interaction devices, like the Kinect camera, projector and touch-screen.

The PO robot, being a simpler robot, has only one computer:

- The Navigation Computer (aka PC-Nav). This computer is running Linux OS (Ubuntu), and is responsible for navigation, localization, system control and actuation of some low-level interaction devices, like LEDs.

The PO robots will not have moving parts, like the head and the arms and also will not include the Kinect Camera, Projector and the touch-screen monitor.

An interaction board is present in both types of robots, but the hardware to move the arms and head is not included in the PO robot. This board will control the LEDs of the mouth, eyes, cheeks and the ones installed in the base (facing the floor). Both robots will have capacitive sensors, but the PO will have only on the head and shoulders. The RFID antenna will be present and powered by the interaction board.

Table 1 describes the devices present in both robots and the computer to which they are connect.

	SO robots		PO robots
	PC-NAV	PC-HRI	PC-NAV
Navigation Computer	▨		▨
HRI Computer		▨	
Sensor&Management board:	▨		▨
batteries	▨ 4		▨ 3
4 batteries chargers	▨		▨
1 Temperature and humidity sensor	▨		▨
4 Bumpers	▨		▨
4 ground sensors (analog IR sensors)	▨		▨
1 Sonar board with 12 sonars	▨		▨
Motors board:	▨		▨
4 motors with encoders	▨		▨
4 motor and driver temperature sensors	▨		▨
7 cooling fans that can be controlled	▨		▨
IMU sensor	▨		▨

2 x Laser Range Finder (LRF), front and rear, 5m range	▨		▨
2 x KIO UWB sensors (optional), front and rear	▨		▨
ASUS Xtion PRO LIVE	▨		▨
Buffalo N600 Gigabit dual band wireless router	▨	▨	▨
RFID Reader	▨		▨
Audio amplifier and speakers	▨		▨
Interaction Board:	▨		▨
1 Mouth Led Matrix	▨		▨
6 RGB controlled devices	▨		▨
2 Eyes (left and right independent)	▨		▨
1 Cheeks (left and right dependent)	▨		▨
3 Under the base (left, front and right LED segments)	▨		▨
Capacitive sensors	▨		▨
2 Shoulders(left and right)	▨		▨
1 Head	▨		▨
2 Arms/hands(left and right)	▨		
1 Rotating head	▨		
4 Arms' limit switches	▨		
Arms Herkulex Control	▨		
Lilliput 10.1'' LED touch monitor		▨	
Kinect Camera		▨	
AAXA P300 Pico projector		▨	
Stargazer (optional)	▨		▨

Table 1: device distribution in the SO and PO robots.

To ease the integration of the different devices with the Linux/Windows computers, each device was connected to the same port in all MOnarCH robots. Fig. 3 depicts the use of the motherboard back panel. Four extra USB ports can be used by connecting USB extensions directly to the motherboard. Fig. 4 shows the position of the 4 extra USB channels. FUSB30 can be used to connect 2 USB3.0/USB2.0 and F_USB1 can be used to connect 2 USB2.0 devices.

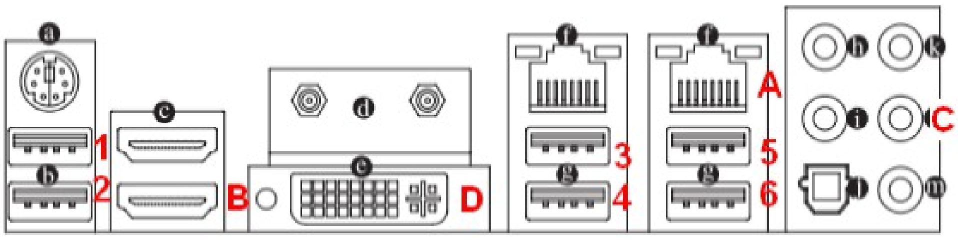

Fig. 3: motherboard Back Panel Connectors

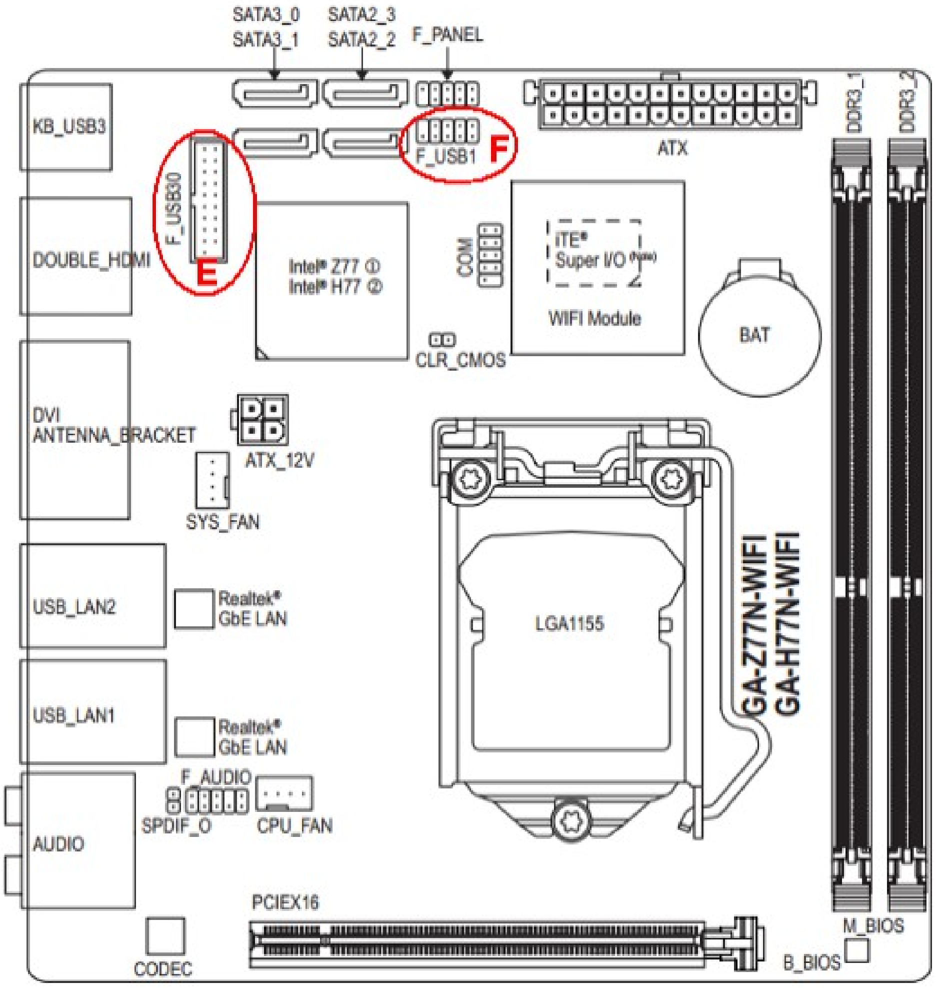

Figure 4: motherboard USB extra connections

The way the devices are connected to the ports of the motherboard is now listed.

<u>SO robots</u>

1. PC-NAV
   - 1 - USB auxiliary extension to be connected to a USB HUB that connects the RFID, the Arms (board4) and the Interaction Board (board3);

- 2 - ASUS Xtion PRO LIVE;
- 3 - IMU;
- 4 - Front LRF;
- 5 - StarGazer or USB auxiliary extension;
- 6 - Rear LRF;
- A - Ethernet connection to the Router;
- B - Not in use;
- C - audio amplifier;
- D - Not in use;
- E1 – Rear UWB (board6);
- E2 – Front UWB (board5);
- F1 - Sensor&Management Board (board1);
- F2 - Motors Board (board2).

2. PC-HRI

- 1 – not used;
- 2 – not used;
- 3 – not used;
- 4 – not used;
- 5 – Touch Screen;
- 6 - Kinect;
- A – Ethernet connection to the Router;
- B – Projector;
- C – Not used;
- D – Monitor;
- E – Not used;
- F – Not used;

<u>PO robots</u>

1. PC-NAV

- 1 – USB auxiliary extension to connect to a USB HUB that connects to the RFID reader and the Interaction board (board3);
- 2 – ASUS Xtion PRO LIVE;
- 3 – IMU;

- 4 – Front LRF;
- 5 – StarGazer or Interaction board (board3);
- 6 – Rear LRF;
- A – Ethernet connection to the Router;
- B – Not in use;
- C – Audio amplifier;
- D – Not in use;
- E1 – Rear UWB (board6);
- E2 – Front UWB (board5);
- F1 – Sensor&Management Board (board1);
- F2 – Motors Board (board2).

2.3. MOnarCH Sensors

The robot is equipped with perception, navigation, interaction, environment and low-level safety sensors. For **navigation** the robot uses encoders to control the velocity of the motors, an inertial sensor to determine the orientation and lasers to detect obstacles and the geometry of the environment. For **perception** and **interaction**, the robot will use a depth camera for people tracking, face analysis and body gesture recognition, microphones and also a RFID reader to identify people carrying a RFID ID card. For **environmental** sensing the robot will be equipped with temperature and humidity sensors. Finally, the bumpers and sonar sensors will provide **low-level safety** sensing. To increase the robustness of localization, some other sensors/solutions are also being evaluated, e.g., StarGazer and UWB.

The onboard sensors are now listed.

2.3.1. Navigation Sensors

The robot will navigate in the environment while making a fusion of measures provided by different sensors. The robot will be able to use a depth camera, two laser range finders, encoders and an IMU sensor to estimate its position and orientation. For obstacle avoidance, mapping and localization it can use the lasers and sonar sensors.

- Inertial Sensor IMU: MPU6050

  Function: Orientation estimation

  Position on Robot Platform: robot's centre of rotation

- Front and rear 2D laser range-finder: Hokuyo URG-04LX-UG01

  Function: mapping, localization and obstacle avoidance

  Position on Robot Platform: one in the front (see Fig. 5, left picture) and another in the rear (see Fig. 5, right picture) of the robot platform base

- Sonar Sensors: Maxbotix EZ4

  Function: obstacle detection (e.g.: glass wall or objects)

Position on Robot Platform: ring of 12 sonars around the robot. Fig. 6 shows the ring of sonars around the robot body.

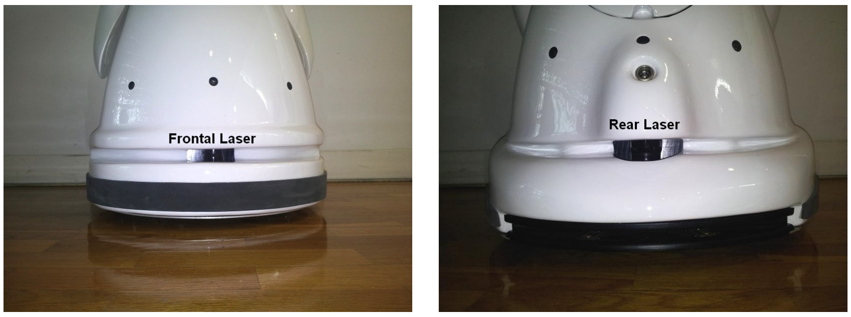

Figure 5: MOnarCH LRFs. Left picture: front LRF. Right picture: rear LRF

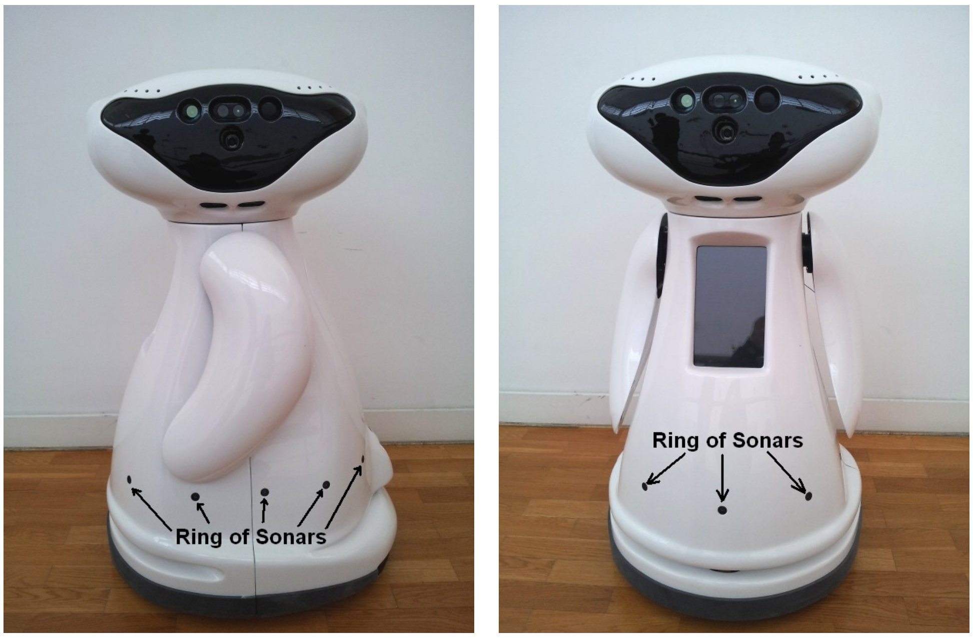

Figure 6: MOnarCH robot ring of sonars

- Depth camera: Asus Xtion Pro Live

  Function: obstacle detection and space geometry analysis

  Position on Robot Platform: top of the robot pointing to the floor. Fig. 7 shows the Asus Xtion mounted on the robot head.

- Stargazer (optional): Hagisonic Stargazer

  Function: Robot Localization

  Position on Robot Platform: Top of the robot, looking up. Fig. 8 shows the Stargazer position on the robot head.

- UWB sensor (optional): Eliko KIO

  Function: robot localization estimation

  Position on Robot Platform: inside the robot, one sensor board in the front and another in the rear of the platform base.

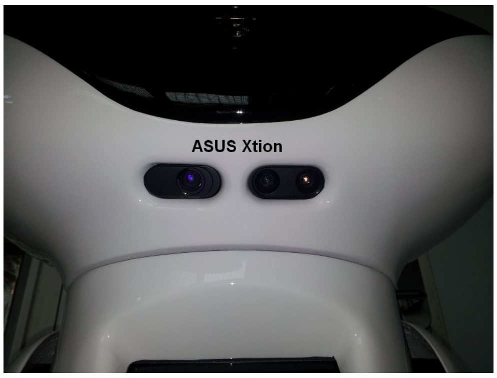

Figure 7: location of ASUS Xtion. Looking down (approx. 45 degs)

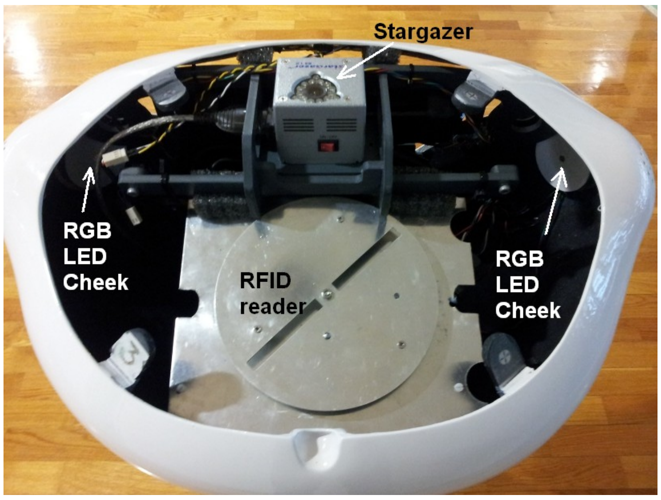

Figure 8: MOnarCH front head devices

2.3.2. Perception and Interaction Sensors

The robot will make use of a depth camera for people detection and sense visual user feedback for natural user interaction. It can also be used to detect changes in the surrounding environment. The perception sensors are the following.

- Depth camera: Microsoft Kinect

  Function: interaction, people detection, gesture recognition

  Position on Robot Platform: top and looking ahead (see Fig. 11).

- Microphone array: Microsoft Kinect

  Function: user sound feedback for natural user interaction

  Position on Robot Platform: turned to the users

- 10.1” Touchscreen: Lilliput FA1012-NP/C/T

  Function: user feedback on specific contents

  Position on Robot Platform: turned to the user

- Capacitive sensors: Sparkfun MPR121

  Function: user feedback on specific points

  Position on Robot Platform: under the shell, one in each arm, one in each shoulder and one in the head (see Fig. 9).

- RFID: RFID integrated reader

  Function: People and objects detection

  Position on Robot Platform: inside the robot head. Fig. 8 shows the RFID position on the robot head

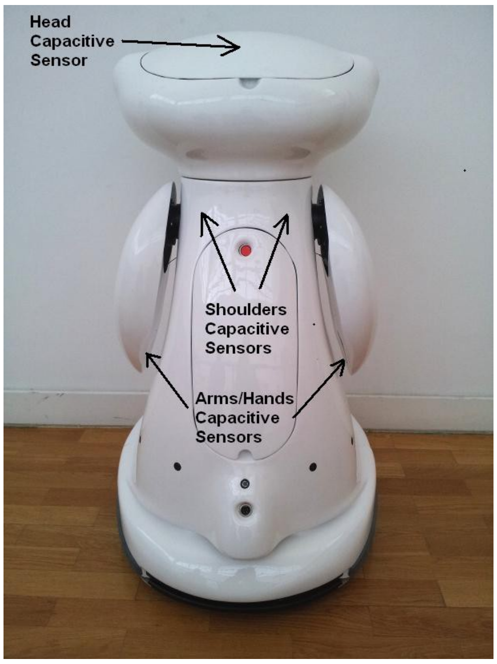

Figure 9 : MOnarCH robot capacitive sensors

2.3.3. Environmental Sensors

The environmental sensors are used to detect environment variations that can affect the normal operation of the robot. These sensors are: temperature sensor and humidity sensors.

2.3.4. Low-level Safety Sensors

The fundamental sensors for low-level safety are the sonar sensors, internal temperature sensors, motor current sensing and the bumper ring switches.

2.4. MOnarCH Actuators

The robot will be equipped with locomotion and interaction actuators.

2.4.1. Locomotion Actuators

For locomotion, this omnidirectional platform uses four motors to drive the four Mecanum wheels.

- Four Maxon RE 35 90W 15V motor with a Maxon GP 32 HP 14:1 Gearbox and encoder HEDS 5540 with 500 pulses

  Function: provide a omnidirectional locomotion system to the robot

  Position on Robot Platform: Inside the platform base, connected to the drive system (see Fig. 10)

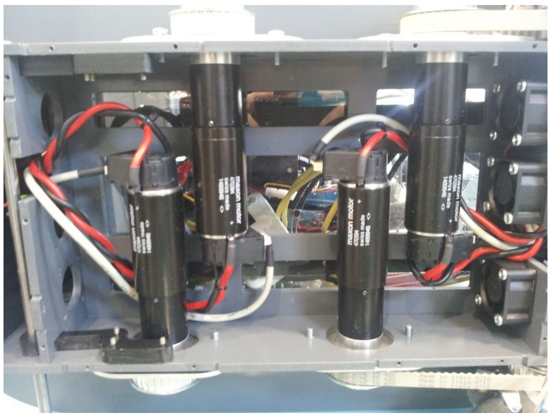

Figure 10: MOnarCH four motors

2.4.2. Interaction Actuators

Here follows the list of interaction devices. The robot is able to display the contents on the interaction monitor or project them over a surface.

- 10.1'' Monitor with Touchscreen: Lilliput FA1012-NP/C/T

  Function: interaction with displayed contents (e.g., AR contents)

  Position on Robot Platform: front of the robot

- Video Projector: AAXA P300 pico projector

  Function: projection of contents

  Position on Robot Platform: projecting to the front of the robot. Fig. 11 shows the position of the projector.

- Arms' motors: Herkulex motors

  Function: Human robot interaction

  Position on Robot Platform: attached to the robot shoulder. Fig. 12 shows the motor and the gears where the arm is attached.

- Head motor: Motor with gearbox and encoder

  Function: Human robot interaction

  Position on Robot Platform: mounted on the robot neck to rotate the head (see Fig. 13)

- Body RGB LED lights

  Function: show robot expressions and movement actions

  Position on Robot Platform: two independent eyes, two cheeks, three LED strips on the robot base (one in the front and one at each side). Figs. 14 and 15 show the RGB LEDs mounted on the robot.

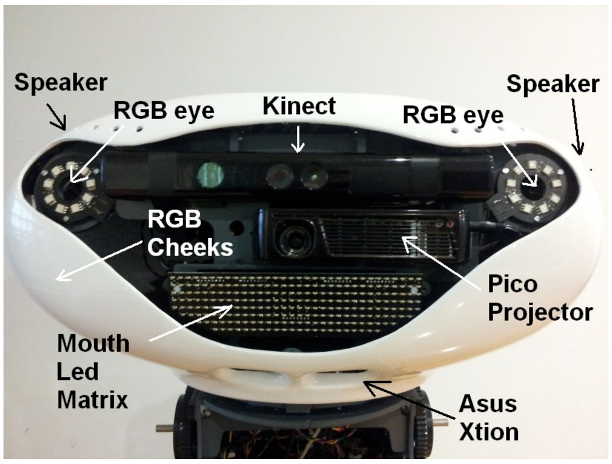

Figure 11: MOnarCH front head devices

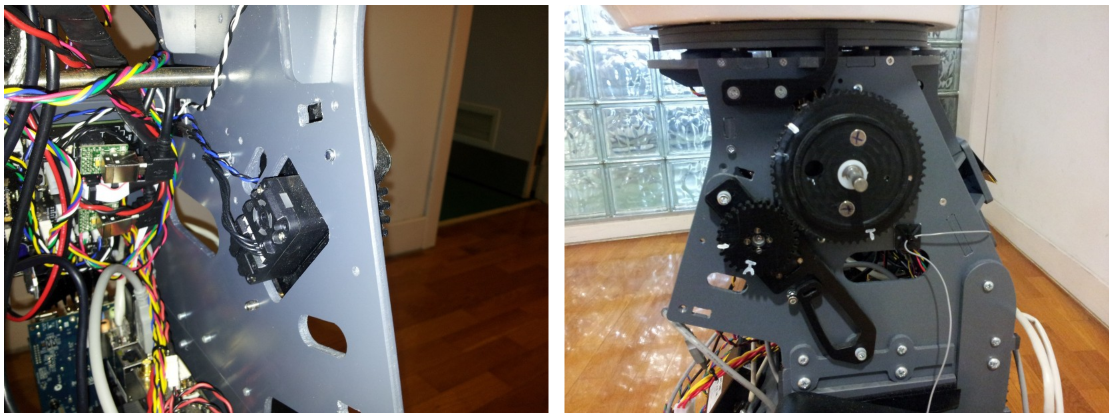

Figure 12: robot shoulder. Left image: Herculex motor. Right image: coupling gear

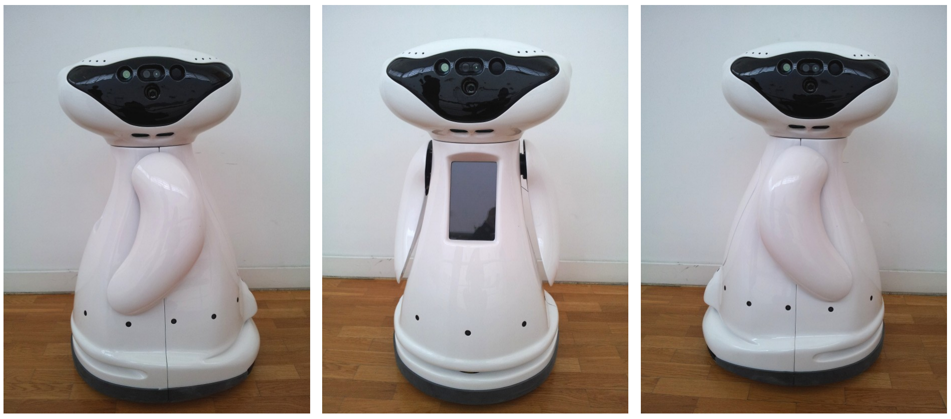

Figure 13: MOnarCH robot rotating head

- Mouth LED Matrix

  Function: show robot expressions

  Position on Robot Platform: on the mouth of the robot. Fig. 14 depicts two possible mouth expressions.

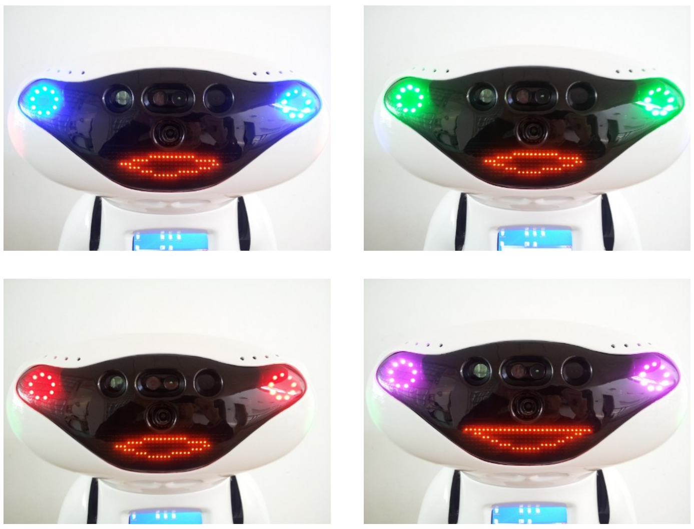

Figure 14: MOnarCH RGB LED eyes and mouth dot-matrix LEDs

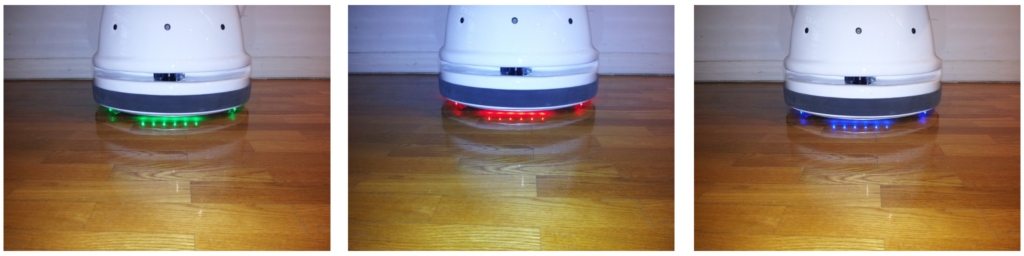

Figure 15: MOnarCH RGB Base LEDs

- Stereo Speakers

  Function: sound exhibition of contents; robot communication

  Position on Robot Platform: on the robots' head (see Fig. 16)

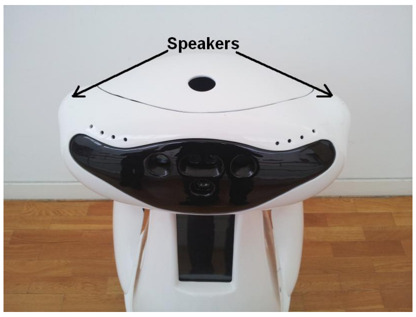

Figure 16: MOnarCH speakers' location

3. MOnarCH Robot Design and Mechanics

At the beginning of the project there was no clear idea about the type of robot(s) to be used in the project.

The DoW mentioned the use of two types of robots: one robot targeting social interactions with people/kids and another robot with a primary goal of acting as an active sensing unit.

On these basis it was asked to the different partners a list of equipment that should be included/integrated in each kind of robot.

This chapter describes the process of development of the MOnarCH robot. Starting from a list of equipment and storyboard scenarios, until the real functional robot with the required capabilities and appearance.

The structure of this Chapter is as follows:

- The Storyboards → How they shaped the robots design;
- Early Stage Drawings of the Platform Base → Decision of the type of locomotion, power and navigation equipment;
- High-level devices → The detection and interaction devices selected by the partners;
- The first and second input design of the shell → How the platform and devices shaped the robot design;
- Final drawing of the inner platform base and its construction → From the scratch to the real world. The robot platform bases are built and tested by the partners;
- Design of the upper body inner structure → Taking in account the detection and the interaction high-level devices to be integrated, the upper body inner structure is designed;
- Final shell design → Having the complete inner structure defined and designed, it was possible to finish the shell design;
- Preparation and production of the MOnarCH shell → Step-by-step preparation and building of the robot shell;
- The final result → The shell is mounted on the robots and tested.

3.1. The Storyboards

The MOnarCH project focus is on **social robotics** using networked heterogeneous robots and sensors to interact with children, staff, and visitors, **engaging** in edutainment activities in the pediatric infirmary at the Portuguese Oncology Institute at Lisbon (IPOL). Fig. 17 shows the map of the IPOL building.

The following general issues were pointed as critical for the development of the robots:

- The physical characteristics of the robots should be adequate to the envisioned operation;
- There should be a clear definition of interfaces in-between the different components, internal and external;
- Dependability: availability, reliability, security, and safety;
- Ethics related with the operation scenario.

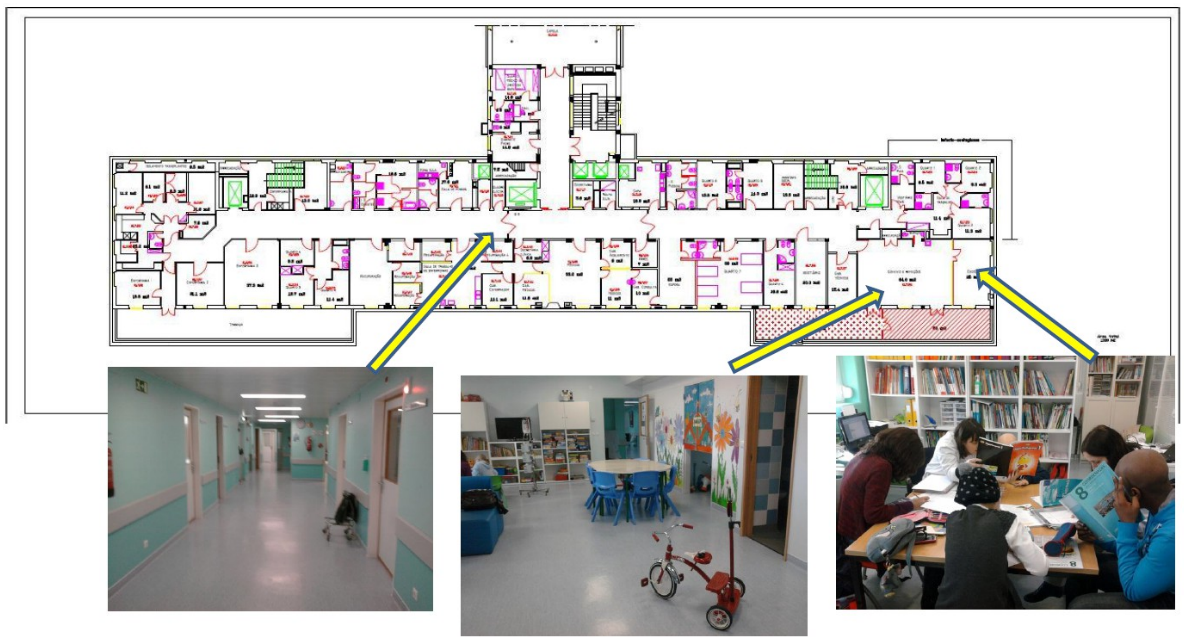

Figure 17: IPOL map and main spaces of operation

Meanwhile the partners were studying and refining the three storyboards included in the DoW. For each storyboard subsets of operational areas have been selected:

- School teacher - One playroom, one corridor and one class room.
- Joyful warden – One playroom with a bicycle garage, one corridor, one room and one forbidden area.
- Interactive game – One playroom and one corridor.

An interesting point of view as also given by the kids at IPOL. Fig. 18 depicts some drawings from a survey on how children see generic robots.

The way each robot should address each storyboard allowed to define the constraints on the robot design:

- The range of allowable linear and angular velocities
- The body volume of the full robot
- Aesthetics
- Weight of the platform
- Payload
- Power supply autonomy
- Self-safety features
- Human-oriented safety features
- Cost

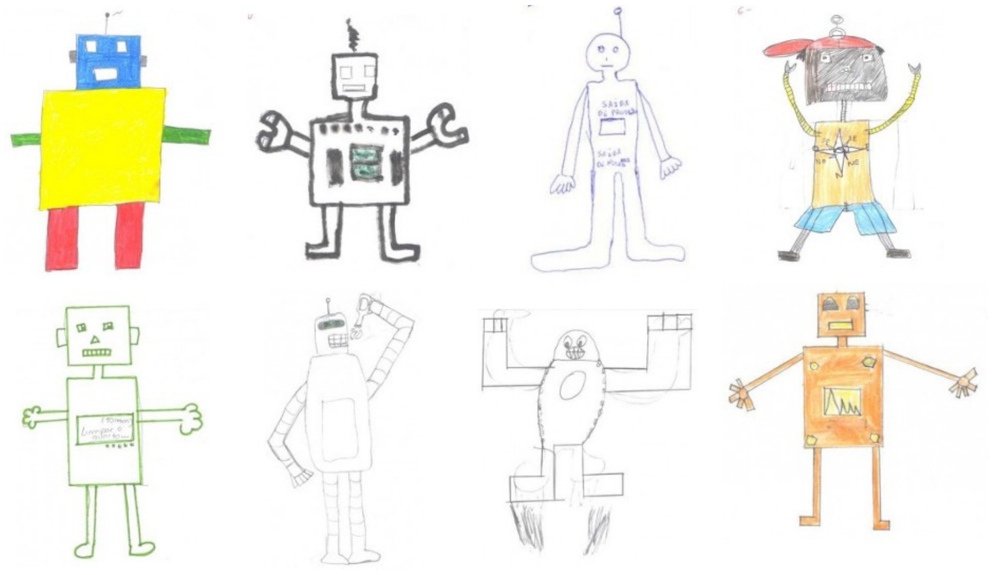

Figure 18: robots as seen by children.

During this initial phase of the project some conceptual robot appearances drawings were also presented by YDR to the partners. Some examples are presented in Fig. 19.

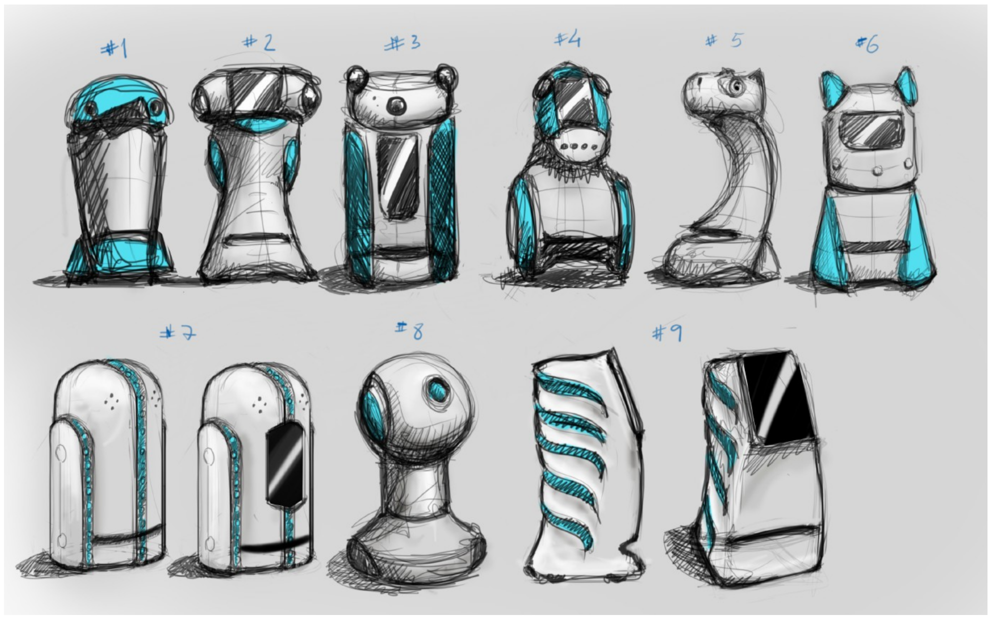

Figure 19: robot appearance conceptual proposals

3.2. Early Stage Drawings of the platform Base

In the early stages of the project, the consortium was still considering the use of a differential two-wheeled robot platform. As the user case scenarios were being defined and the constraints posed by the environment of operation were discussed by the Consortium, it became evident that the mobility capability of the robots could be a critical issue to the achievement of project goals. Consequently, IDM opted for developing and building an omnidirectional robot platform.

The early stage design of the inner platform considered the project specifications regarding autonomy, navigation and kinematics:

- 4 x 12V 17-20Ah batteries (~1KWh)
- 4 x 12V 90W DC motors
- Schematics: Power board + Motor drivers board
- 4 x Omnidirectional Mecanum wheels (see Fig. 20)
- Front Laser

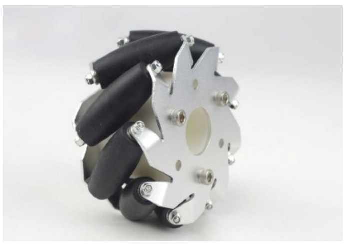
Figure 20: Mecanum wheel

These specifications were used to start the design of the MOnarCH platform base inner structure. Fig. 21 depicts the first design of the MOnarCH robot inner platform base structure inside a conceptual shell base, allowing to have an idea of volume and already the required cut on the shell for the laser range finder beam.

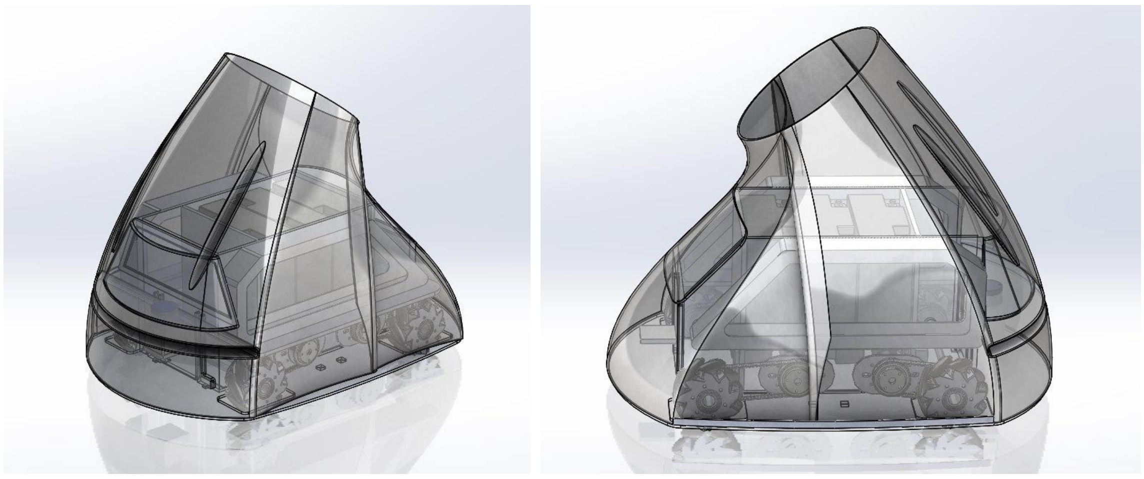
Figure 21: early stage drawings of the platform base

3.3. High-level Devices

After the definition of the low-level platform components, the partners started to study the needed devices to perform the storyboards. This produced a list of high-level devices to include in the robot platform:

- Two depth cameras with microphone (Kinect type)
- Three servo motors to actuate two robot arms and a head
- One 10" touch-screen
- One pico-projector

- Audio amplifier with speakers
- LEDs on the robot body
- Capacitive cells on the robot body
- Optional devices: RFID reader, Stargazer and UWB

3.3.1. Robot Appearance (1st Iteration)

This definition of the high-level devices was followed by a first iteration on the design of the robot external appearance. Figs. 22 to 25 show this first iteration on the design of the shell where different external appearances were still being considered.

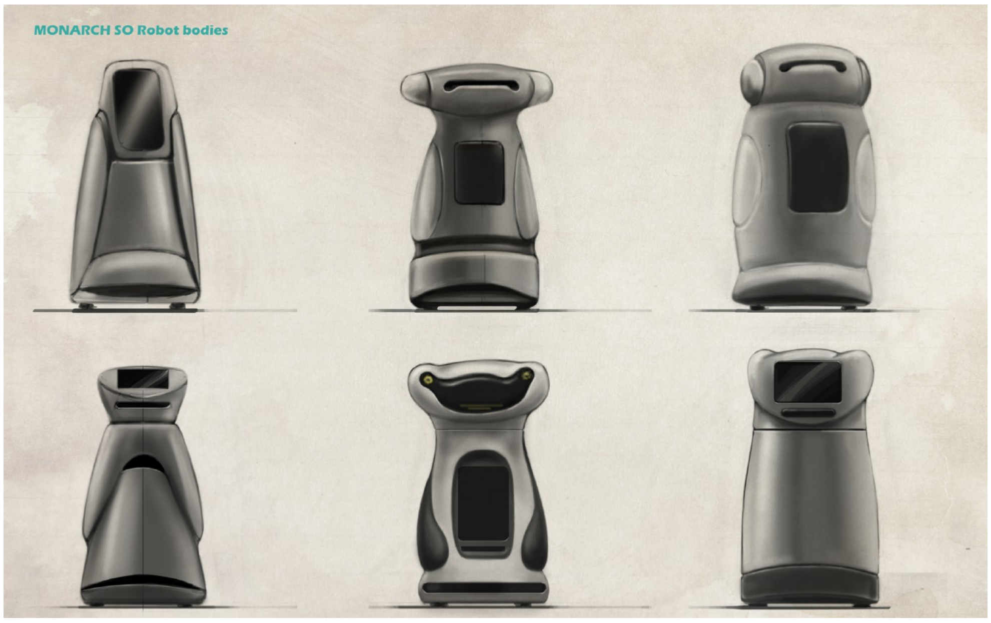

Figure 22: body appearance proposals

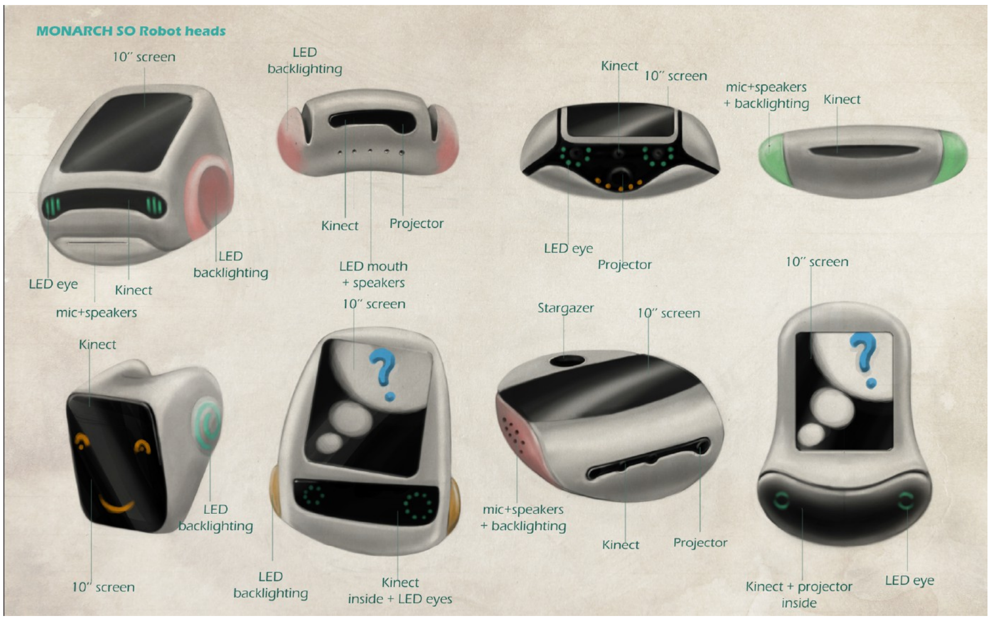

Figure 23: head appearance proposals

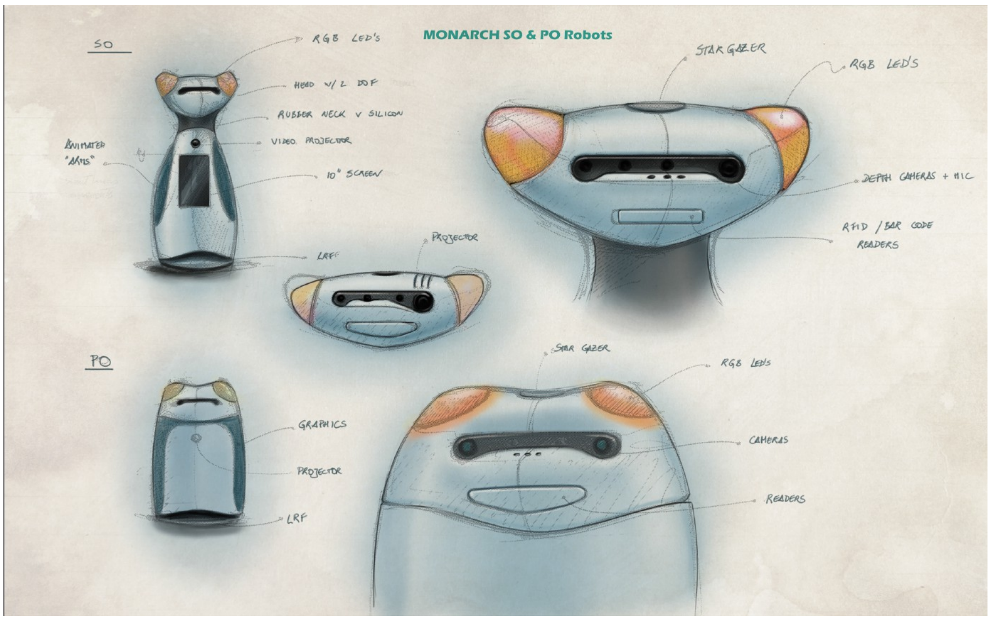

Figure 24: study of body appearance

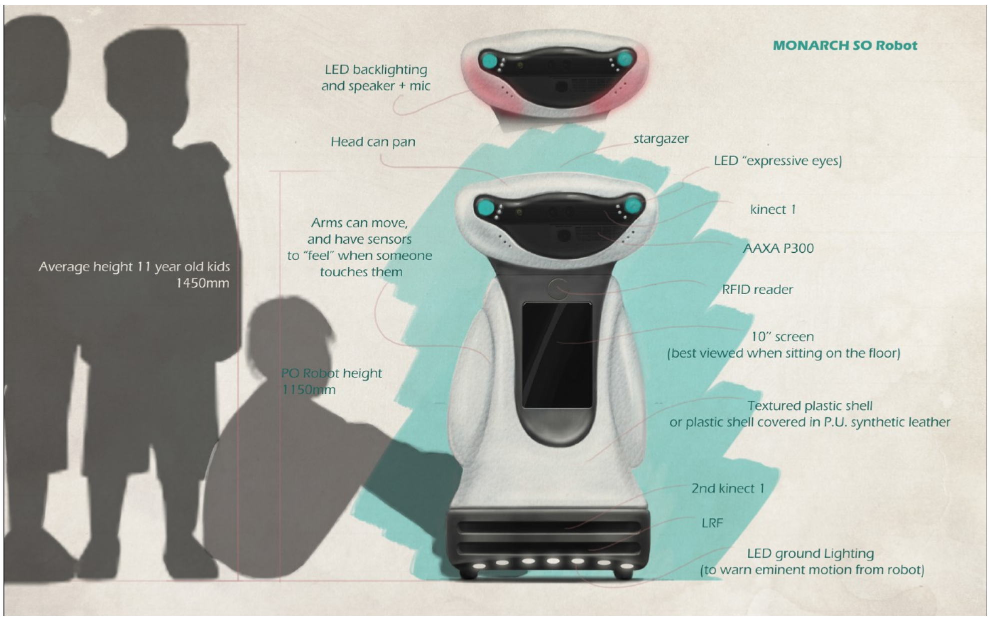
MONARCH SO Robot
LED backlighting and speaker + mic
Head can pan
stargazer
LED "expressive eyes)
Arms can move, and have sensors to "feel" when someone touches them
kinect 1
AAXA P300
RFID reader
Average height 11 year old kids 1450mm
10" screen (best viewed when sitting on the floor)
PO Robot height 1150mm
Textured plastic shell or plastic shell covered in P.U. synthetic leather
2nd kinect 1
LRF
LED ground Lighting (to warn eminent motion from robot)

Figure 25: defined robot appearance (1st iteration)

3.4. Platform Base Production

The design of the platform base was then concluded. Fig. 26 shows the platform base CAD rendering, including the four 100mm Mecanum wheels, the motors, the battery distribution, the bumpers, the sonars, the docking mechanics and the front laser ranger finder. This platform can accommodate up to four standard 12V-20Ah lead-acid batteries (close to 1 KWh of power). In one of the sides of the platform is possible to see the Motor Board and on the other side it is possible to see the Sensor&Management board.

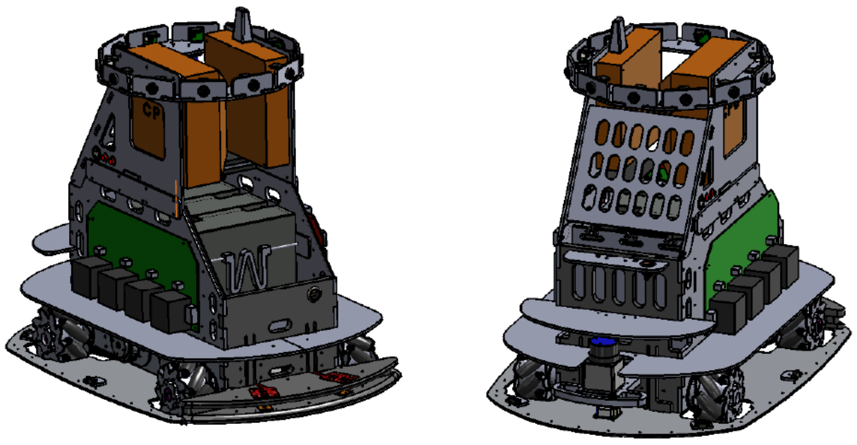

Figure 26: platform base CAD rendering

The platform base prototype was concluded in month 12. The main body of this platform is made of polyoxymethylene (POM) and Rigid PVC, which are materials with high stiffness, low friction and excellent dimensional stability. Fig. 27 depicts the assembled robot base.

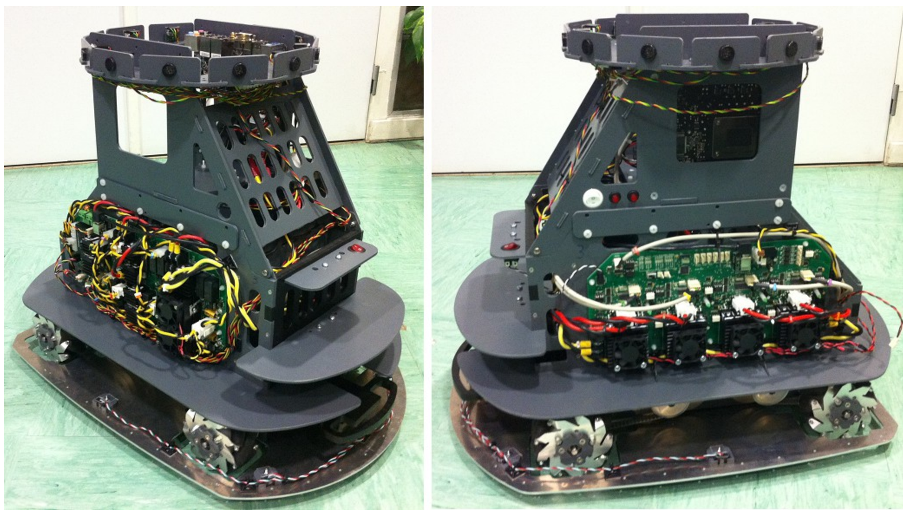

Figure 27: assembled MOnarCH robot platform base

The materials used on the construction of the platform base are:

- Body: Polyacetal - POM (PolyOxyMethylene) 10 mm thick plates; rigid PVC 4 and 6 mm; and transparent polycarbonate 2mm
- Motor fixings: aluminium 4mm
- Main plate base: aluminium 3mm
- Transmission gears type/material: metric toothed aluminium pulley T5 with 32th
- Belt drive type/material: Endlose Polyurethan-Zahnriemen

All the materials, except the aluminium ones, were cutted in-house using IDM's CNC machine. The parts were cleaned and assembled, then all the devices and cables were installed.

3.5. Upper Body Development

3.5.1. Robot Appearance (2nd Iteration)

In parallel with the inner structure construction there was a second iteration on the design of the robot external appearance (see Fig. 28).

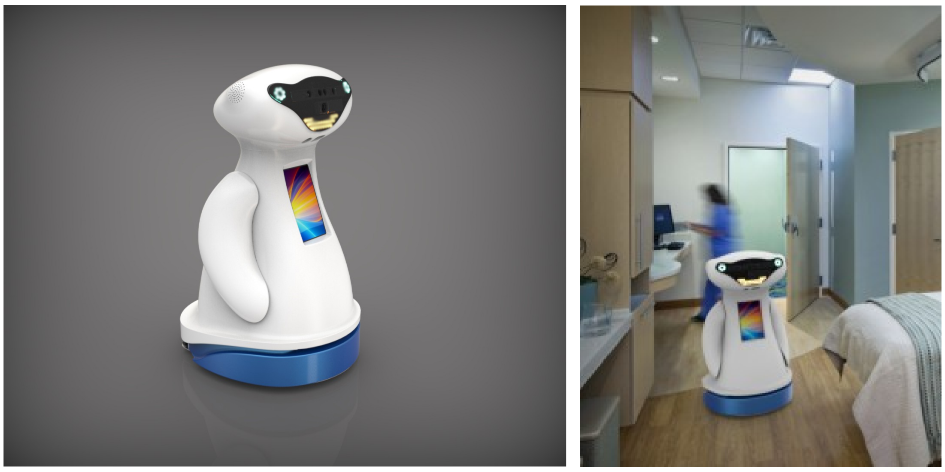

Figure 28: robot appearance (2nd iteration)

This iteration was already considering the location of the different onboard devices specified by partners as depicted in Figs. 29 and 30.

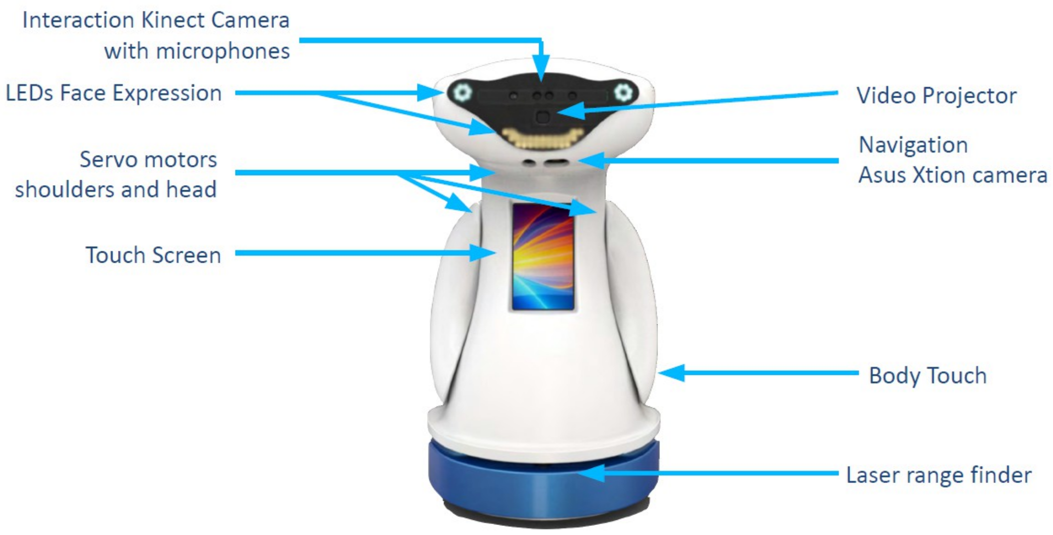
Interaction Kinect Camera with microphones
LEDs Face Expression
Servo motors shoulders and head
Touch Screen
Video Projector
Navigation Asus Xtion camera
Body Touch
Laser range finder

Figure 29: location of onboard devices

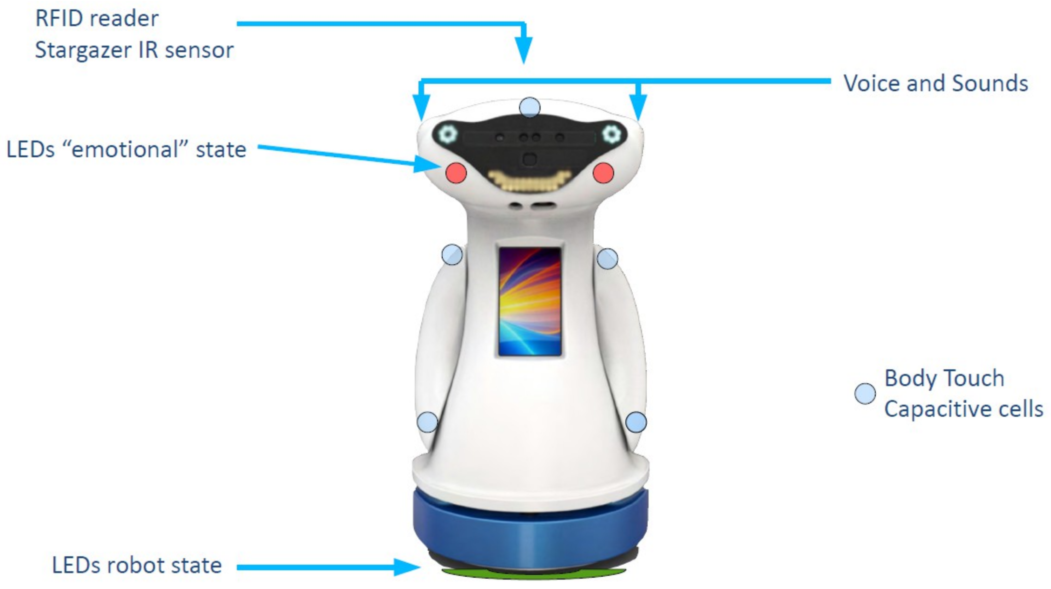

Figure 30: location of onboard devices (cont.)

3.5.2. Conclusion of Shell Design

Fig. 31 shows the final design of the shell. This final design resulted from close collaboration between IDM and YDR. The position of the inner structure and devices was tested against the outer shell design, this allowed both partners to make the required corrections on the inner structure and the shape of the shell.

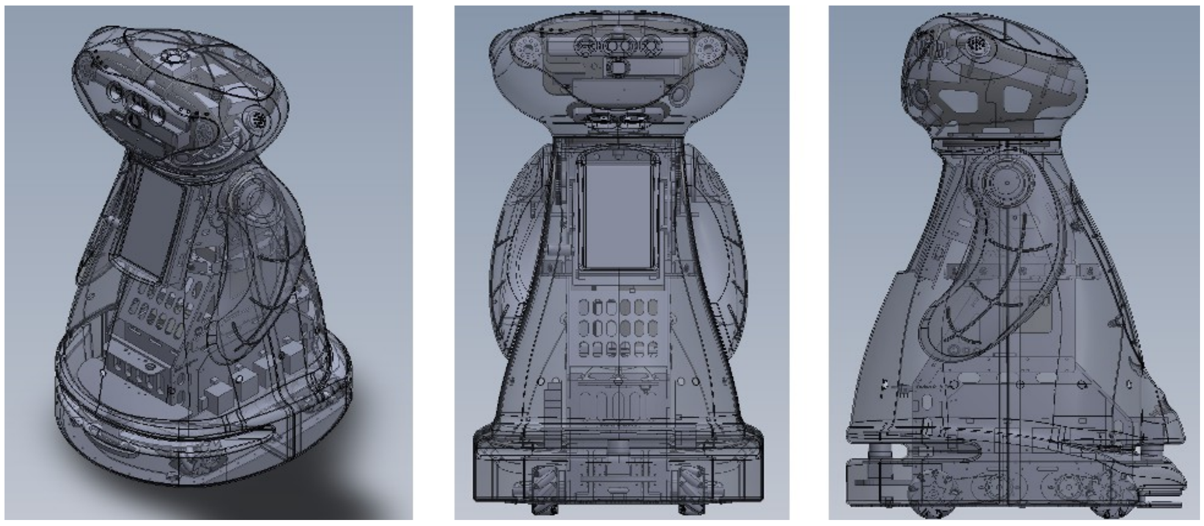

Figure 31: robot inner body with outer shell

3.5.3. Upper Body Production

Based on the final design of the shell and on specified location of the different devices, the design of the upper body of the robot was then concluded. Fig. 32 shows the final mechanical design just before starting the production of the entire structure.

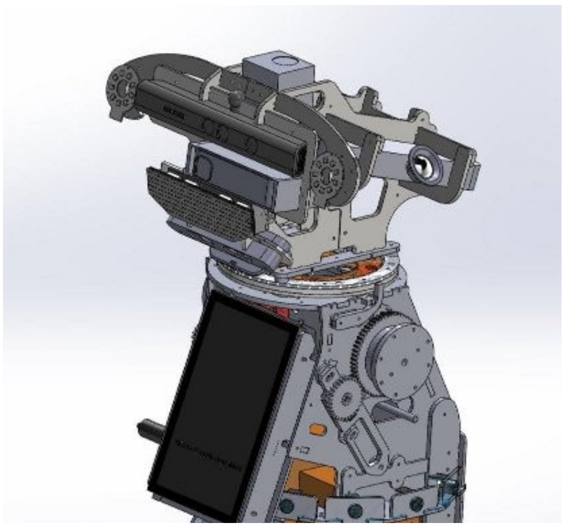

Figure 32: MOnarCH robot upper body mechanical design

The following materials were used on the construction of the upper body and arms transmissions:

- Upper Body: Rigid PVC 4 and 6 mm; and transparent polycarbonate 2mm;
- Arms: Polyacetal - POM (PolyOxyMethylene) 10 mm and 20mm thick plates.

All the upper structures were cutted in-house using IDM's CNC machine. Fig. 33 depicts the upper body assembly and installation of devices.

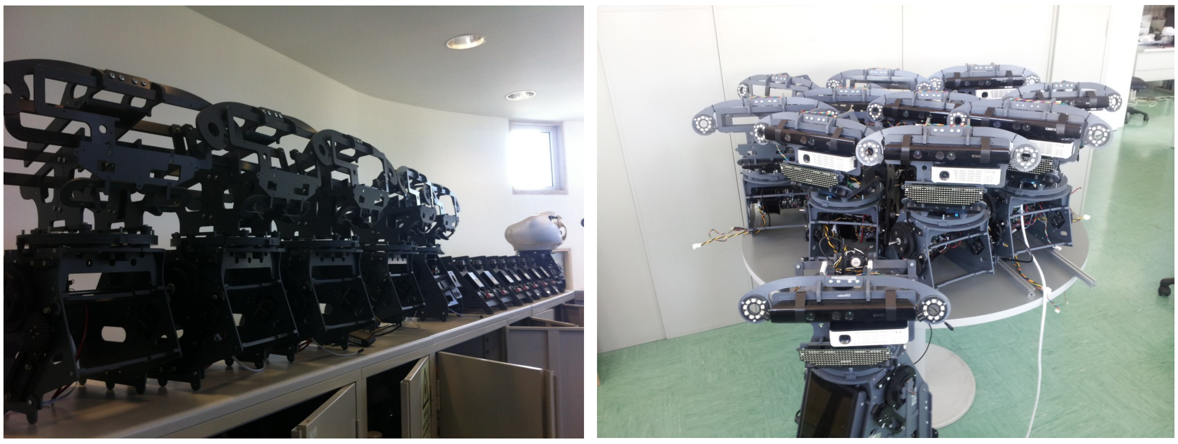

Figure 33: MOnarCH robot upper structure assembly and devices integration

To conclude the assembly process of the MOnarCH robots inner body, the upper structure with the installed devices was mounted over the platform base. Fig. 34 shows MOnarCH robots already with the full inner body structure with high-level devices already installed.

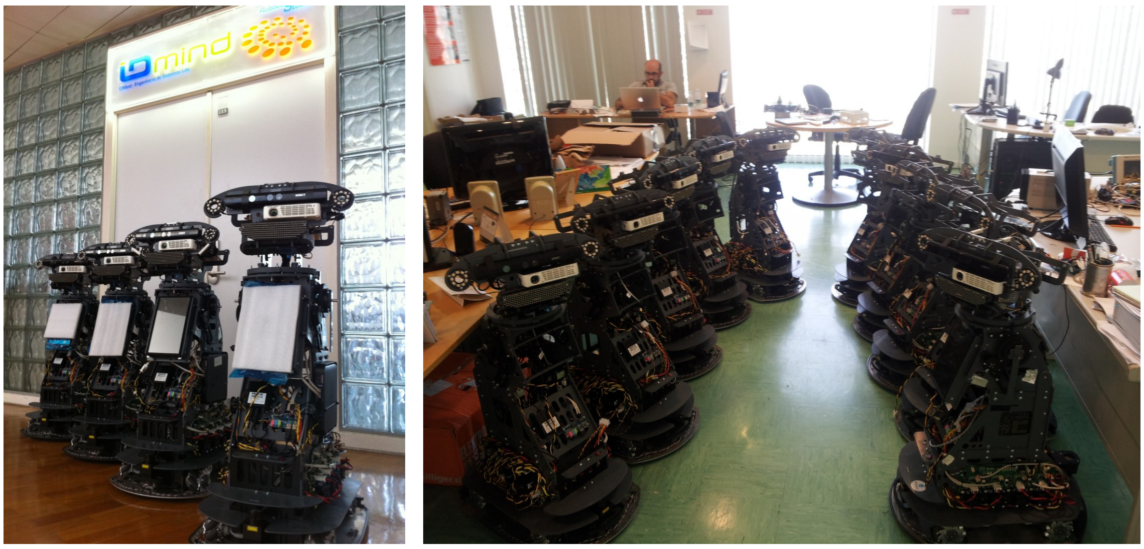
Figure 34: MOnarCH robots platforms

3.6. Outer Shell Production

The external design of the shell had to be prepared for production. This included the division of the body shell into parts taking in consideration the envisioned production methods, the tools, the materials and the maintenance and testing constraints. Figure 35 depicts this stage of the process.

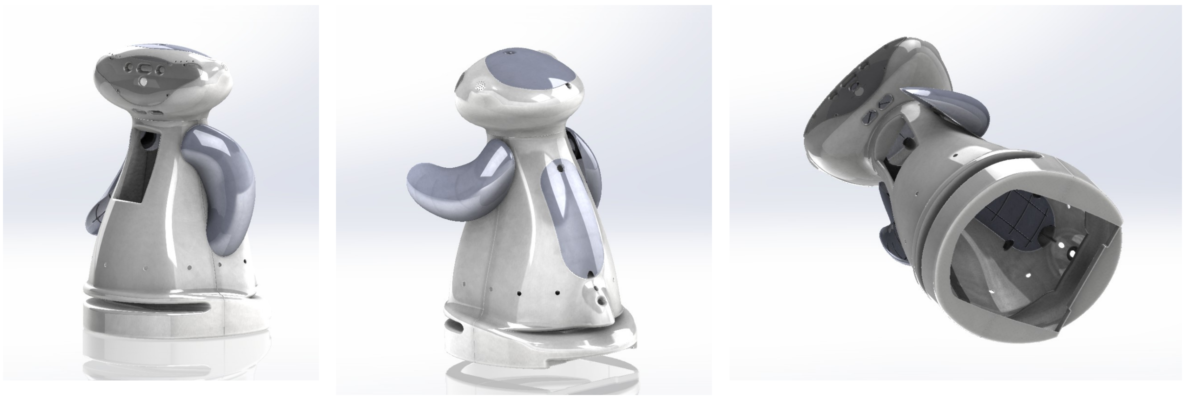
Figure 35: design of the shell for production

The production of the different shell parts required the production of a mould. The production of the mould was made in-house through the use of IDM's CNC machine to drill different blocks of roofmate (see Fig. 36, left) that were later putted together to create an inner body 3D mould (see Fig.37). Some smaller parts of the mould were also produced in-house through the use of a 3D printer (see Fig. 36, right).

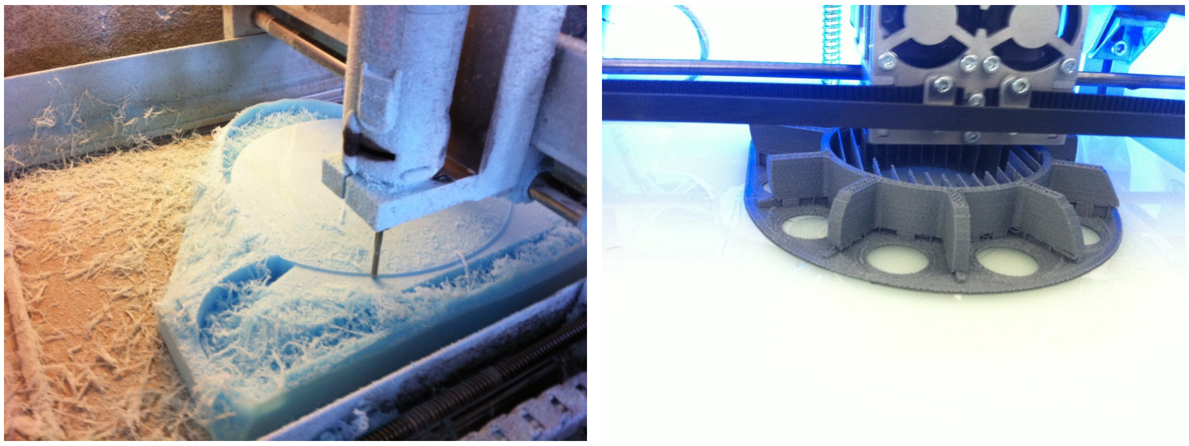
Figure 36: production of the mould. Left image: drilling machine. Right image: 3D printer

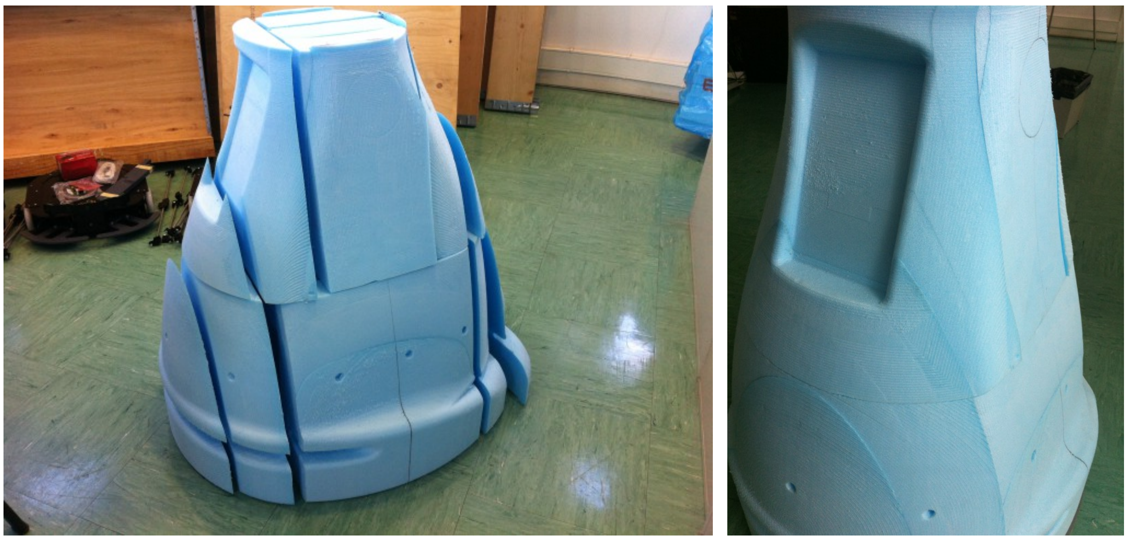
Figure 37: putting the drilled parts together for the body mould

The same process was repeated to create the head mould (see Fig. 38).

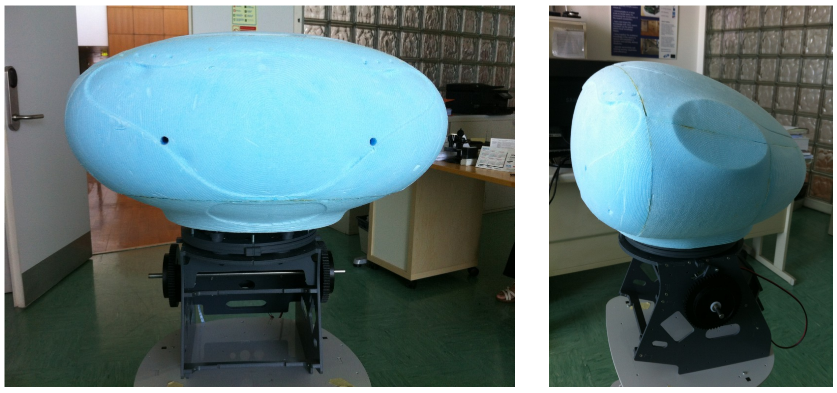
Figure 38: head mould.

The moulds were delivered to a fibreglass manufacturer that used them to create the different shell parts. Fig. 39 shows the unpainted fibreglass shells during the production process and Fig. 40 shows the first time that the first shell was tested on the robot structure.

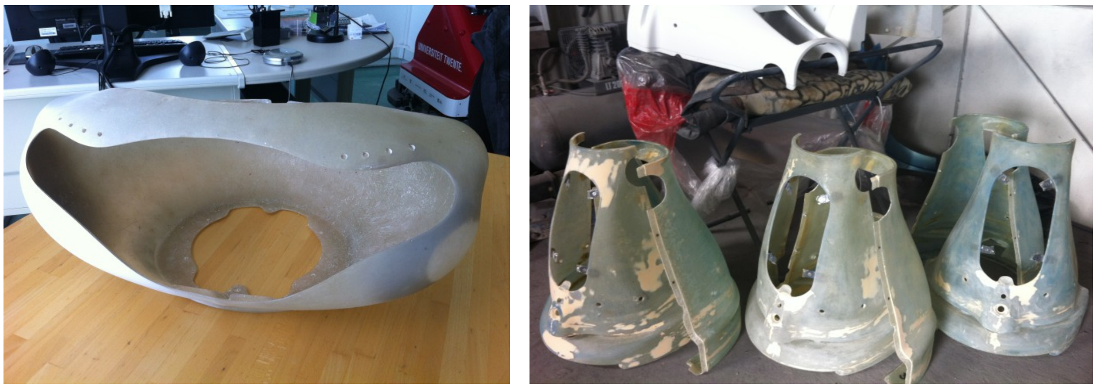
Figure 39: production of the shell

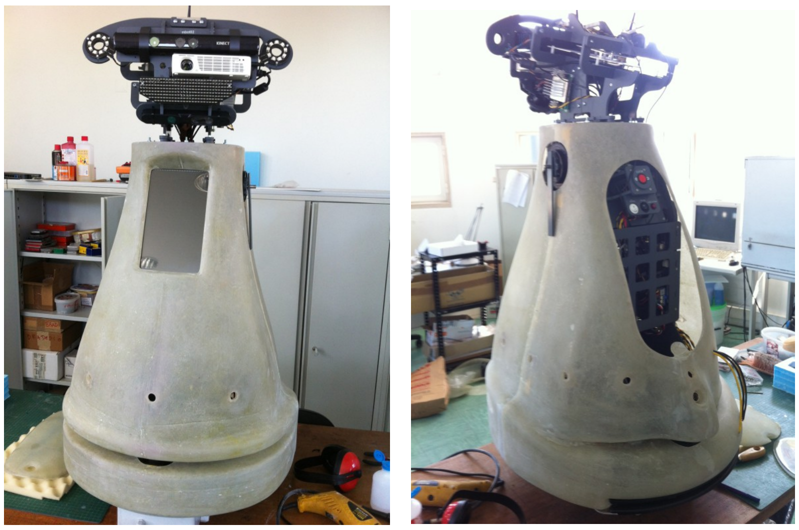
Figure 40: testing the shell with the inner body

After the validation of the shells on the robots, the produced fibreglass shells were sent for painting (see Fig. 41).

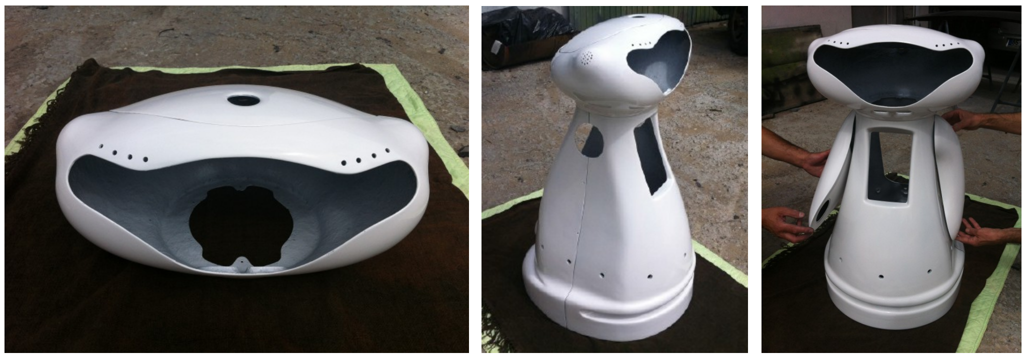
Figure 41: painting the shell

To finish the robot look a visor for the face was produced. For the visor IDM made use of a vacuum forming process.

The process started by machining a visor mould in roofmate. Then a negative mould of the roofmate visor was produced using soft plaster (see Fig. 42). After dry, this negative mould of soft plaster was used to create a positive mould in hard plaster (see Fig. 43). Finally, this hard plaster mould was then used as mould in the vacuum forming process. To give a final dark appearance, the inside of the plastic visor was painted with a dark colour (see Fig. 44).

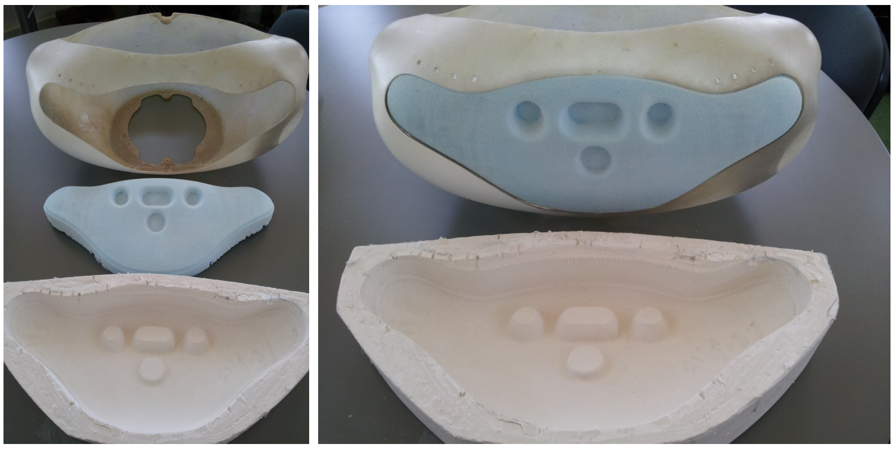
Figure 42: making the mould for the visor (i)

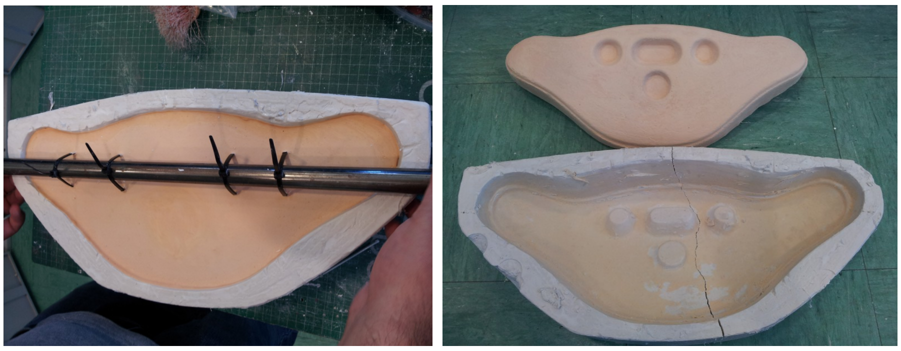
Figure 43: making the mould for the visor (ii)

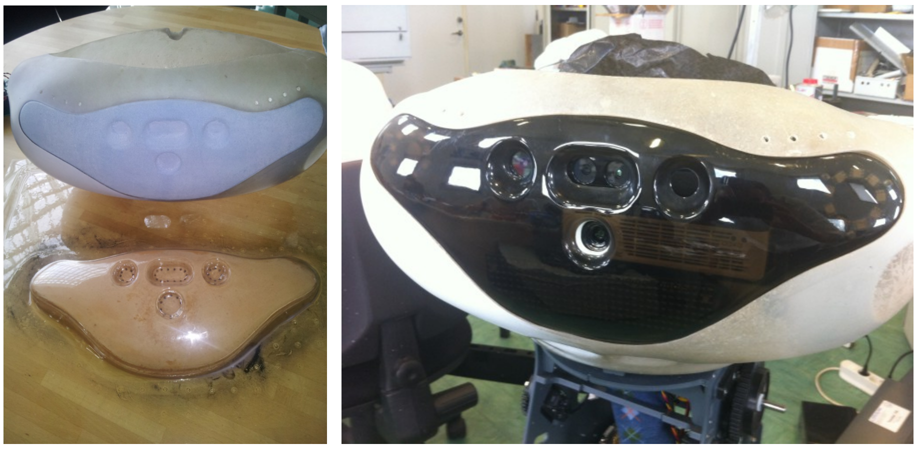
Figure 44: visor production

3.7. The Final Result

The final result was the creation of two types of robots. Fig. 45 depicts a complete SO robot. The PO robot is depicted in Fig. 46.

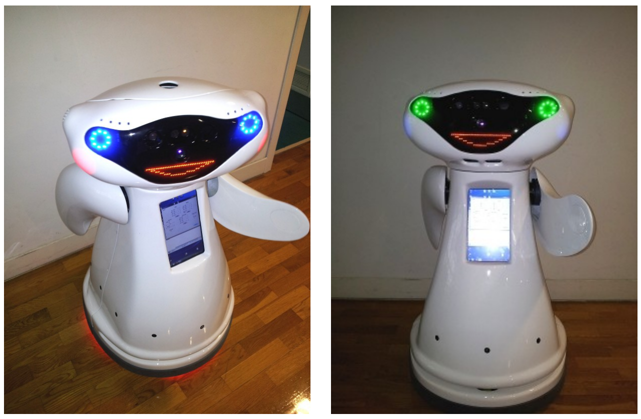

Figure 45: fully assembled MOnarCH's SO robots.

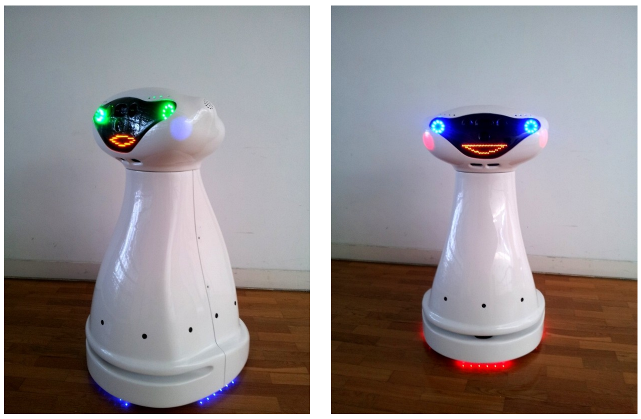

Figure 46: fully assembled MOnarCH's PO robots.

4. MOnarCH Robot Architecture

This section describes the power and communication architecture implemented in the MOnarCH robot.

The MOnarCH robot will be an autonomous robot able to navigate in indoor environments, with no human intervention. To power all the components, it will use up to 4 LiFePO4 with PCM 12V (17Ah) batteries. The first subsection explains the power distribution inside the MOnarCH robot.

The second subsection shows the low-level communication architecture. The electronics that control the motors, read the sensors and actuate the motors are considered to be low-level electronics.

Finally the third subsection presents the high-level communication architecture. Each computer will connect to different devices, such as monitors, cameras, projectors and electronic boards, through the use of different technologies such as USB, HDMI or Ethernet connections.

4.1. MOnarCH Electronic Power Architecture

The robot can be powered by several 12V 17-20AH batteries. It uses one 12V battery to deliver power to the motor drivers. And up to 3 other batteries to provide energy to all the other computers and electronic components, each of these batteries will contribute to a pool of energy where the electronic devices can be powered from. An individual charging unit is used inside the robot to charge each battery. The batteries and the power in the robot will be managed by the Sensor&Management Board that measures the battery levels, battery charge, and also controls the units (motors, sensors and actuators) powered by the batteries.

All onboard electronic systems can be powered by the battery system. The ATX computer power supply will provide regulated voltages (from 5V to 12V). Fig. 47 depicts the onboard power architecture. One additional 12V DC-DC converter will also be used to provide the necessary regulated power for Stargazer, touch-screen monitor and the projector.

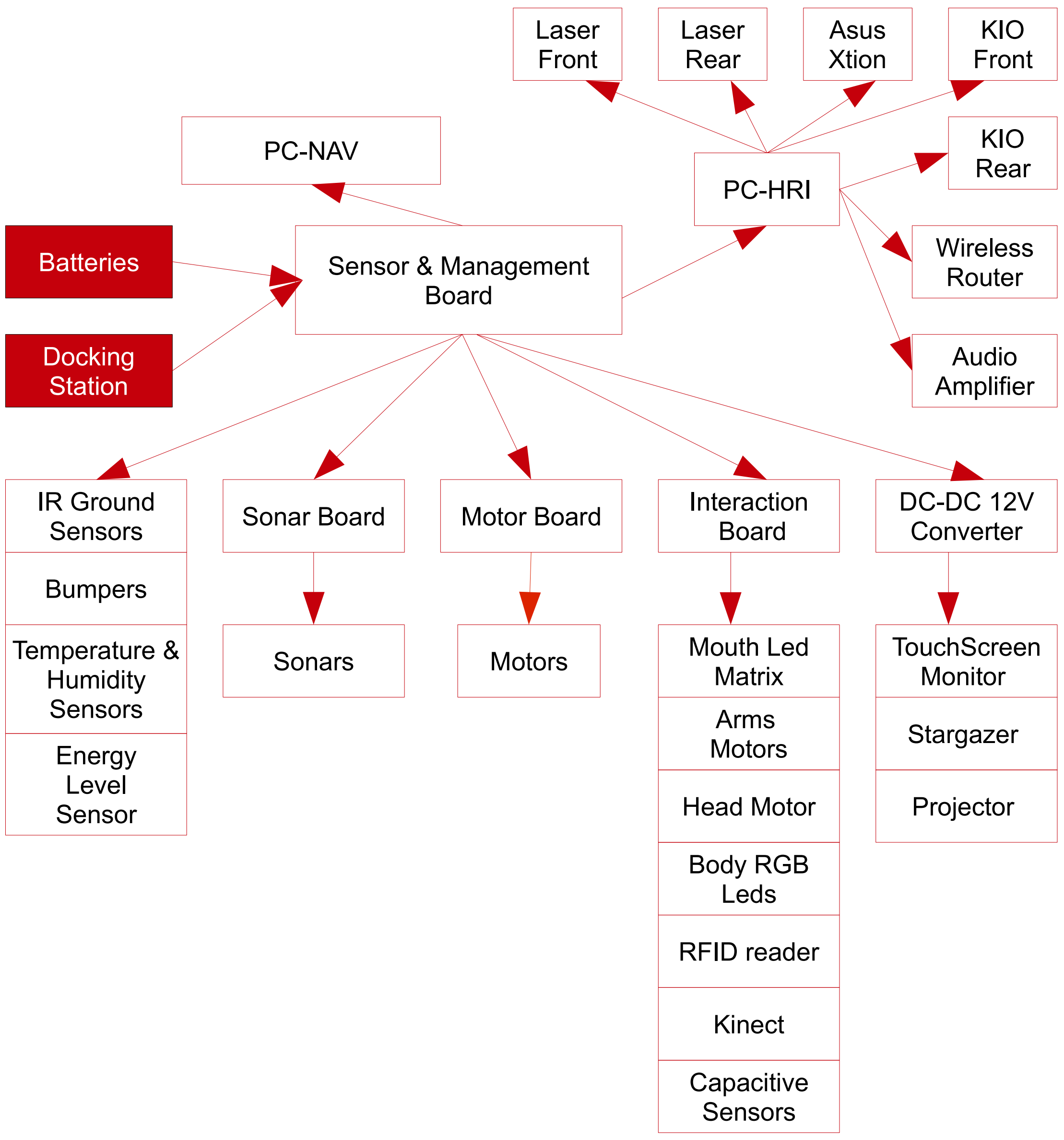

Figure 47: MOnarCH Robot Power Architecture

Table 2 shows the equipment that is connected to the pool of energy and their predicted maximum current and power consumption.

For the SO robot the maximum power consumption is 210W and 110W for the PO robot. Each battery will be able to power up to 240W. This means that the minimum expected working time of a SO robot with the 3 full charged batteries is around 3 hours and for the PO robot with 2 full charged batteries is around 3,5 hours. Tests performed with the robot showed that the robot is able to drive continuously with a full motor battery for more that 5 hours.

In charging mode all the batteries will be put on charge and some of the devices will be powered by 18V from the charging station. The motor drivers will not be connected to the 18V of the power station.

Battery Name	Powered item	Nominal Current (Ah)	Power (Wh)
Motor (4Ah)	12V Motor Driver Power	4.00	48.00
Electronics PC1 and PC2 energy pool (SO 17.3Ah) (PO 9.2Ah)	Navigation Computer	3.50	42.00
	Human Robot Interaction Computer	3.50	42.00
	Buffalo Wireless Router	1.75	21
	Sensor&Management board	0,45	5.4
	Motors board	0,3	3.6
	Interaction Board	0,4	4.8
	Frontal Horizontal Hokuyo	0.21	2.5
	Rear Horizontal Hokuyo	0.21	2.50
	Kinect Camera	0.40	4.80
	Asus Xtion Camera	0.5	2.5
	KIO Front	0.05	0.25
	KIO Rear	0.05	0.25
	Sonar ring	0.10	1.20
	Body Led lights	0.40	4.80
	Mouth Led Matrix	1.5	7.5
	Audio Amplifier	0.5	6
	RFID Reader	0.3	2.7
	Stargazer	0.07	0.84
	Arms Motors	1	12
	Head Motor	0.4	4.8
	Inertial sensor (by the Computer USB port)	0.10	1.20
	TouchScreen Monitor	0.83	10
	Pico-projector	2.00	25.00

Table 2: MOnarCH batteries and maximum power usage.
Note: items in gray are not included in the PO robot

4.2. Low-level Communication Architecture

The onboard PC-NAV computer communicates with the Sensor&Management Board and the Motor Board using 2 USB ports. The onboard PC-HRI computer communicates with the Interaction Board using 2 USB, one that communicates with only with the arms Herkulex motors and the other with rest of the interaction components that are implemented in the interaction board.

In each board there are USB-to-RS232 converters that convert the USB data packages to serial RS232 packages for the boards' controllers.

The Sensor&Management board and the Motors board controller communicate with each other allowing the exchange of information between them. This communication channel can be used to the execution of low-level behaviours, for example, react against an imminent collision, enter into charging mode with motors shut down, reduce the motors' velocity when the batteries are low, or react to changes that can affect the robot's operation, which is fundamental to the improvement of the overall system dependability.

The main controller from the Sensor&Management Board will communicate with other microcontrollers using Inter-Integrated Circuit (I2C) communication ports. The main controller will act as the master and the other microcontrollers will behave like slaves. The Sensor&Management Board will control the battery management and charge, sensor acquisition, devices actuators and the sonar acquisition board. The Motor Controller Board will connect to the PI Motor controllers and also to temperature sensors.

Each controller will have a low-level fault diagnosis that will check the operation state of each microcontroller and also monitor all the communication between the devices. The low-level communication architecture is depicted in Figure 48.

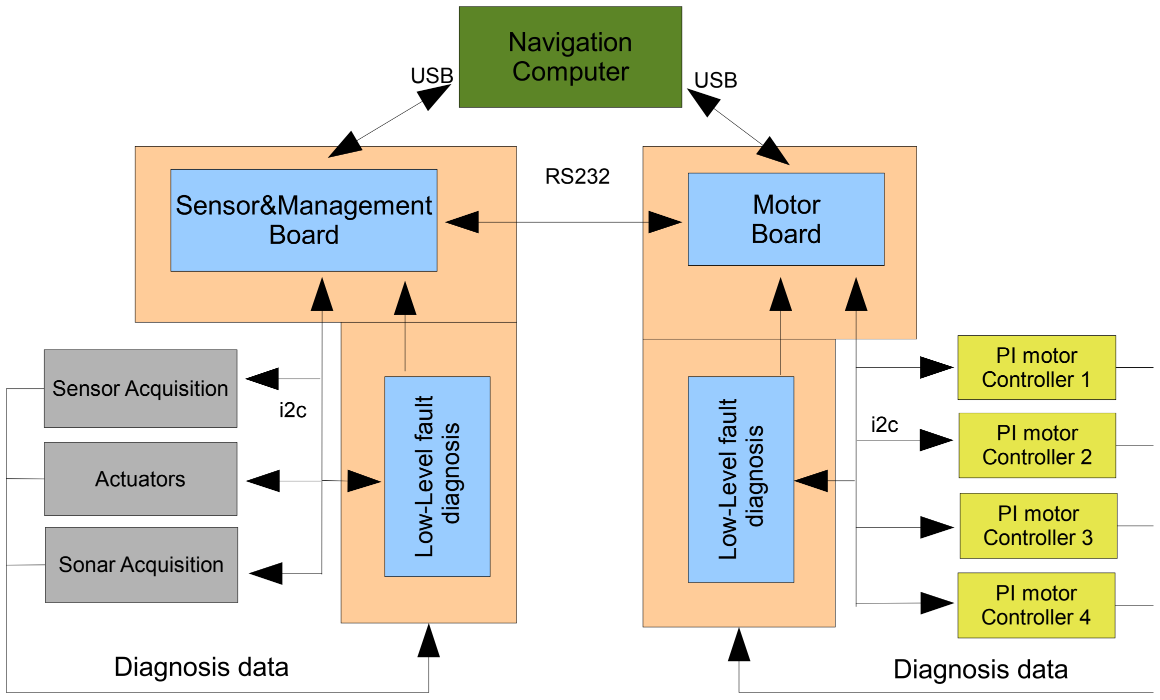

Figure 48: low-level PC-NAV communication architecture

The PC-NAV will communicate with the Interaction Board using two different USB-TO-RS232 converters. Fig. 49 depicts the connection between the PC-NAV, the Interaction Board and the devices controlled by it. The control of the arms uses only an USB-To-RS232 converter and there is no need to add additional micro-controllers to control the motors. The other interaction components need processing. The interaction uses a micro-controller able to receive information from the PC-NAV and process it to be able to generate the expected effects. To control the different devices this micro-controller will use I2C and SPI communication protocols with them.

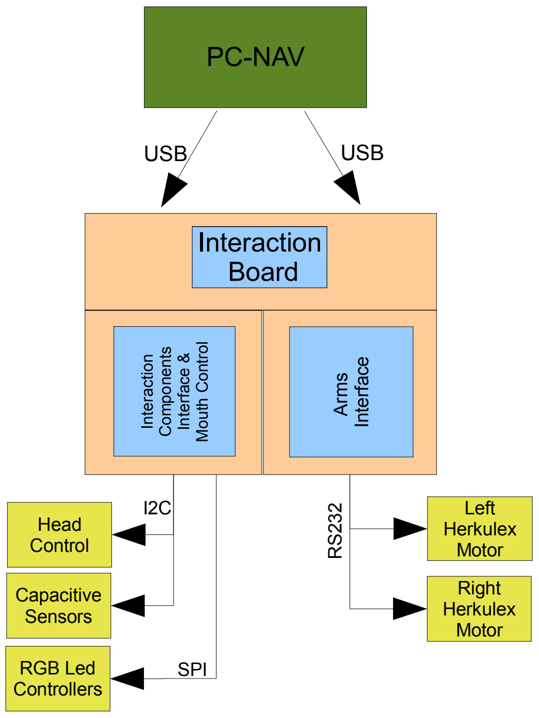

Figure 49: low-level PC-NAV communication with Interaction Board

4.3. High-level Communication Architecture

The MOnarCH robot will connect to a local wireless network through an onboard wireless bridge. Each onboard computer, with its own IP, is connected by cable to a LAN port on the wireless bridge.

The PC-NAV is connected to the navigation sensors and to the platform board controllers using USB ports. The PC-HRI is connect to the projector using a HDMI output, to the monitor using a DVI output, to the Sound System using the audio line out, and will use USB connections to connect to the Kinect camera ant to the touch-screen.

The high-level communication architecture is depicted in Fig. 50.

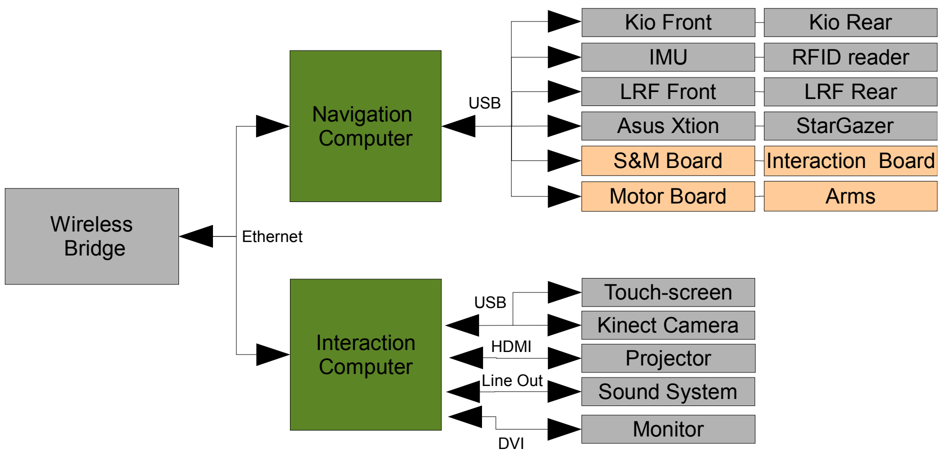

Figure 50: High-level Communication Architecture

5. MOnarCH Robot Electronics

Several electronic boards have been designed and added to the MOnarCH hardware. The following subsections show and describe the boards created for the project.

The following electronic systems have been included on the MOnarCH robot:

- **Motor Controller Board** - this board is responsible for the locomotion. By the use of one microcontroller that communicates with the PC-NAV and sends commands to four PI controllers that control four high power H-Bridges, it is possible to control the motors installed on the platform.
- **Sensor&Management Board** - communicates with the PC-NAV, controls the battery management. It connects, disconnects, measures and charges the onboard batteries, reads the environmental sensors and bumpers and connects to the ground and sonar sensors boards.
- **Sonar Board** - allows the use of sonars in the robot. By sending sonic pulses and by capturing the obstacles' echoes it is possible to detect and measure the distance between the robot and near obstacles.
- **Ground Sensors Board** – allows the use of IR ground sensors to detect changes on the ground colour or detect stairs/holes.
- **IMU Board** – allows the connection of the popular MPU-6050 that combines a MEMS 3-axis gyroscope and a 3-axis accelerometer in the same chip to create a inertial sensor.
- **Interaction Board** - allows the PC-HRI to control the head, arms of the MOnarCH robot, control the body LED lights and also read the capacitive sensors.
- **Charger Docking station** - this is an external device that facilitates the autonomous docking of the robot and the charge of the batteries without human intervention.

5.1. Motor Controller Board

The Motor Controller Board manages the robot locomotion. The Master Motor Controller uses a **PIC18F6527** microcontroller to control and manage all the communication between the high-level robot Navigation Computer and the Slave PI Motor Controllers. The Slave PI Motor Controller uses a **PIC18F2431** microcontroller to provide all of the necessary control signals to the motor drivers. The overall architecture of the Motor Controller Board is depicted in Fig. 51.

The Master Motor Controller will:

- run low-level control loops to check for critical changes in the motor system that can affect the robot operation;
- provide a I2C bus Observer that checks the information received from the I2C Slave devices to understand faults in the communication or on the devices;
- check the good function of each Slave PI Motor Controller by changing status information between them;
- control the motor and drivers' temperature and control the power of fan devices to cool them;
- monitor the electronics, motors and the drivers' power supply.

The Slaves PI Motor Controllers will:

- receive velocity references from the Master Motor Controller;
- periodically read the encoder pulses;
- send the read encoder pulses to the Master Motor Controller;
- calculate the error between the reference velocity and the read velocity;
- implement a PI controller to calculate the motor actuation;
- implement an anti-windup error limit;
- implement a reference velocity acceleration/deceleration profile;
- remove the dead zone of the motor;
- limit the maximum velocity of the motors.

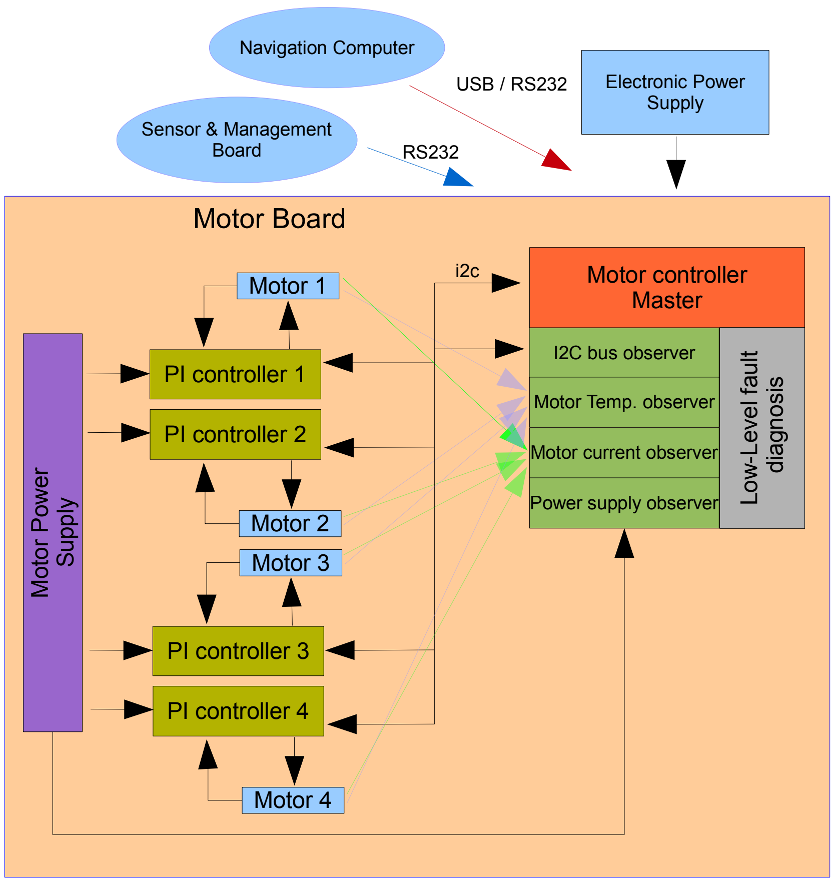

Figure 51: Motor Controller Board Architecture

The Master Motor Controller receives velocity commands from the computer and returns the encoder pulses. The Controller connects to four PI microcontrollers that generate the control actuations to follow velocity references. Each microcontroller connects to the motor using a power H-bridge, and provides the pulses measured by the encoder.

Each microcontroller is optically isolated from the motor driver using a high-speed optocoupler for the control actuation signals and an optical amplifier for the current measurements. It is also optically isolated from the computer communication port, again using a high-speed bidirectional optocoupler.

Several low-level fault diagnostics have been implemented to detect problems in the normal working of each component or lack of communication.

Fig. 52 shows the assembled Motor Controller Board. The board has several information LED lights allowing a visual check of the state of each component previously described.

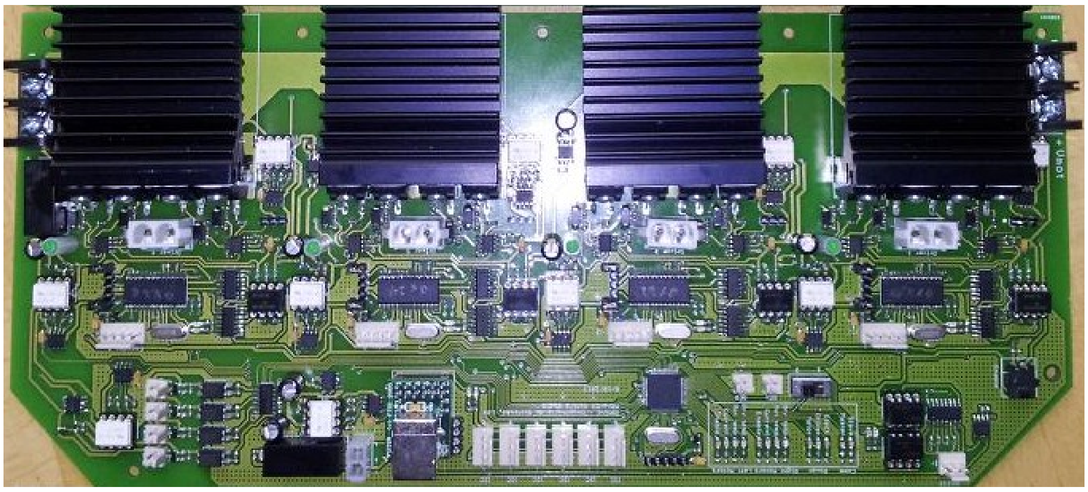

Figure 52: Motor Controller Board

The Motor Board communicates with the Sensor&Management Board exchanging information about the system status, diagnostics and environment condition, allowing low-level robot reactions to changes that can affect its operation.

It is able to communicate with the computer using a USB-to-RS232 converter and with the Sensor&Management Board through a bidirectional RS232 communication.

5.2. Sensor&Management Board

The Sensor&Management Board is responsible for the power management and also the sensor acquisition. It receives orders from the onboard robot PC-NAV and returns information about the batteries, sensors and actuators. The Sensor&Management Board architecture is depicted in Fig. 53.

The Sensor&Management Controller uses a **PIC18F6527** microcontroller to control and manage all the communication between the high-level robot PC-NAV and to control all the actuators and sensors devices connected to the Sensor&Management Board. Fig. 54 shows an assembled Sensor&Management Board.

The Sensor&Management Board is responsible for managing all the power system by:

- measuring the energy level in each battery;
- connecting/disconnecting the power of the devices;
- managing the connection to the Charge Docking station;
- controlling the charge of each battery.

This board is also responsible for connecting a set of sensors and actuators that will be used in the project. Several low-level fault diagnostics have been implemented to detect problems in the normal work of each component and communication. The Sensor&Management Controller will analyse the information gathered from the sensors and will run low-level control loops that will check for critical changes in the environment or system that can affect the robot operation.

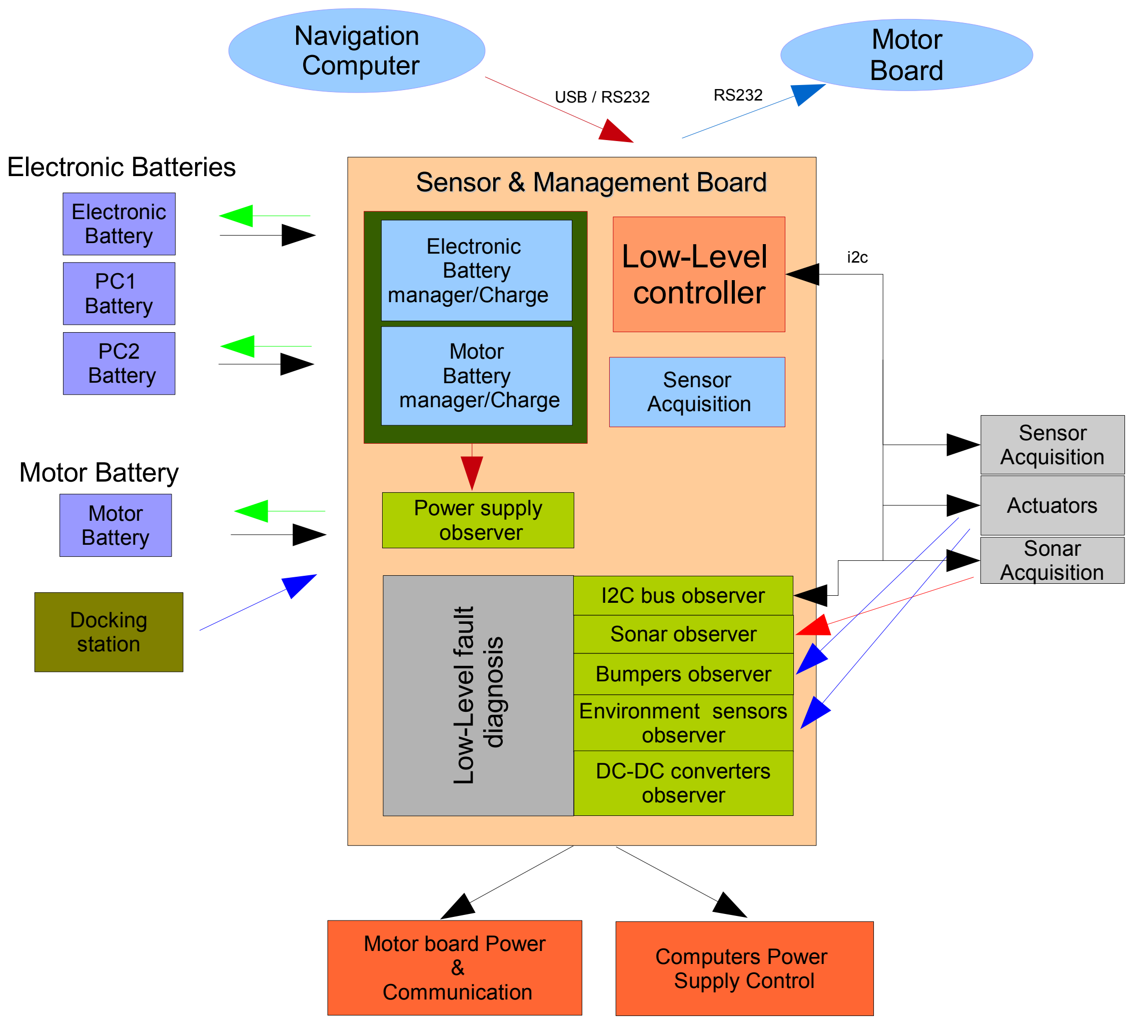

Figure 53: Sensor&Management Board architecture.

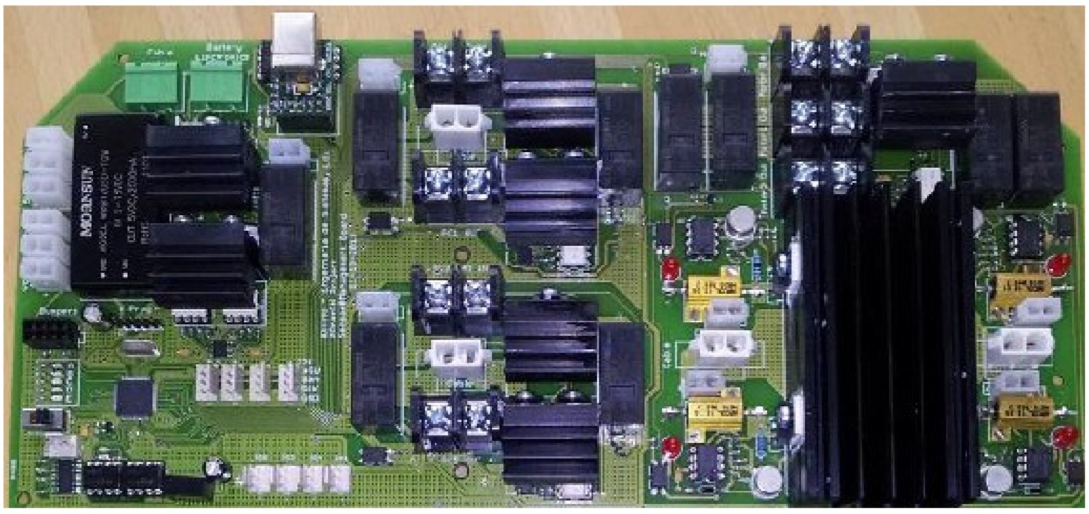

Figure 54: Sensor&Management Board

The Controller has one dedicated channel to communicate with the Motor Controller, allowing it to send direct (and fast) commands to the motors and also to get information from them.

The Sensor&Management Board is able to charge up to 4 batteries independently. Each charging circuit uses a 3-phase lead acid battery charger.

The charger uses the following phases:

1. **Bulk Phase**. In this charging phase the charger will supply a constant 2A current to the battery that is charging. This will allow the battery to charge to about 70% of its capability.
2. **Absorption Phase**. After leaving the Bulk Phase the charger will enter the Absorption Phase where the charger will supply a constant voltage to the battery until it reaches about 100 to 120% of its capability.
3. **Maintaining Phase**. When the battery is fully charged the charger will leave the absorption phase and enter the Maintenance Phase where a 13.4V supply is supplied to the battery until the battery is disconnected from the charger.

The charging process for the 12V 17-20Ah batteries used in the MOnarCH robot is the following. If the battery is dried (9.5V - 0%) the battery will start to charge in Bulk Phase for about 7 hours at 2A. At the end of the 7 hours the level of the battery is about 70%, that means that the charge level will raise about 10% each hour. After +/- 7 hours the charger will enter into the Absorption Phase where there will be an increase of about 5% of level of battery each hour. After 5 to 6 hours the battery level will reach 100%-120% (depending on the type of battery, number of cycles and correct maintenance). After this period the system will enter into the Maintaining Phase, and the battery will maintain the charge level.

Fig. 55 shows the battery charge process using the developed charger. Only the Bulk and Absorption Phases are displayed. The Maintaining Phase start can be set to any point of the Absorption Phase by changing the value of one resistor component.

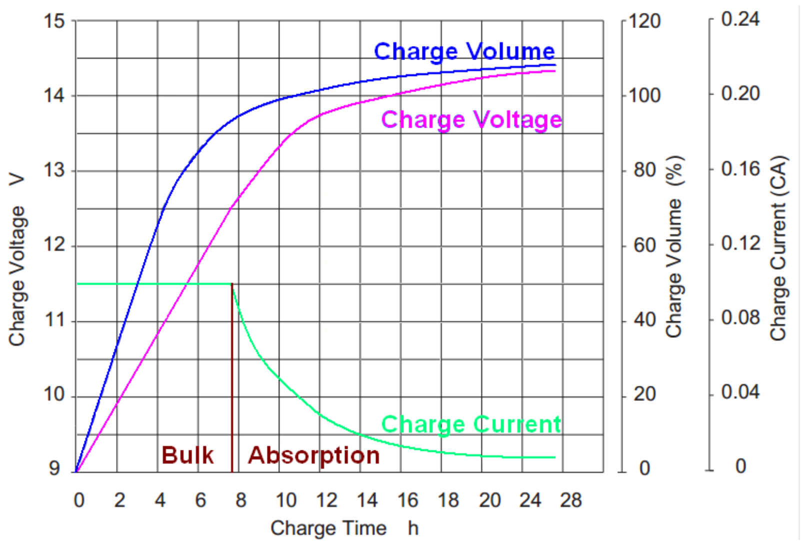

Figure 55: battery charge process

The chargers were designed to charge Lead Acid batteries, but at this moment there is the possibility of changing the Lead Acid to LiFePO4 batteries with PCM protection circuits. These batteries have the same size and capabilities but they weigh ⅓ of the weight of the Lead Acid batteries.

5.3. Sonar Board

The Sonar Board, depicted in Fig. 56, will be used to control the firing of up to 16 sonars. This board is connected to the Sensor&Management Board through an I2C connection. The information provided from this board can be used to create low-level behaviours, for example reducing the velocity of the robot in the presence of an obstacle.

The system uses the Maxbotix EZ4 sonar transducers that are able to detect an obstacle at a distance of up to 4 m.

The board design allows the users to configure the sonar sampling and also to connect or disconnect the sonar readings or power. It can be used to allow the individual fire and reading of all or some of the sonars or the fire of all the sonars at the same time and the individual reading making a radar-like sensor.

The MOnarCH robot uses a ring of 12 sonar. Three groups of 4 sonars were created:

- Group A – includes sonars 1, 4, 7 and 10 (0º, 90º,180º and 270º)
- Group B – includes sonars 2, 5, 8 and 11 (30º, 120º, 210º and 300º)
- Group C – includes sonars 3, 6, 9 and 12 (60º,150º 240º and 330º)

The sonars in the same group are fired at the same time and the distance from the obstacles to each of the sonars is measured. To reduce the crosstalk effect, all the sonars in each group are disposed in 90º multiples.

Each group takes about 100ms to fire and process the echo. The processing of a new group is only started after the end of the processing of the last group (A->B->C->A->B->...).

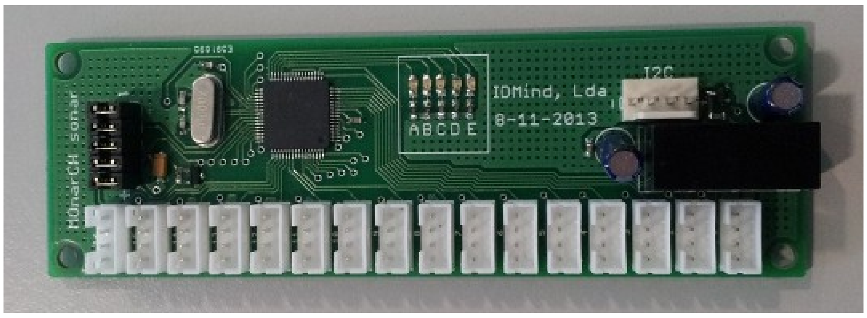

Figure 56: Sonar Processor Board

5.3. Ground Sensors Board

The Ground Sensors board, depicted in Fig. 57, allows the connection of up to 4 IR ground sensors that can be used to detect changes on the floor colour or the beginning of descend stairs. This board is connected to the Sensor&Management Board through an I2C connection. The information provided from this board can be used to create low-level behaviours, for example avoiding the robot from falling down in the stairs.

The MOnarCH robot uses 4 ground sensors: two on the front and one in each side of the robot (see Fig. 58)

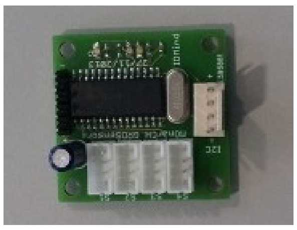

Figure 57: Ground Sensors Board

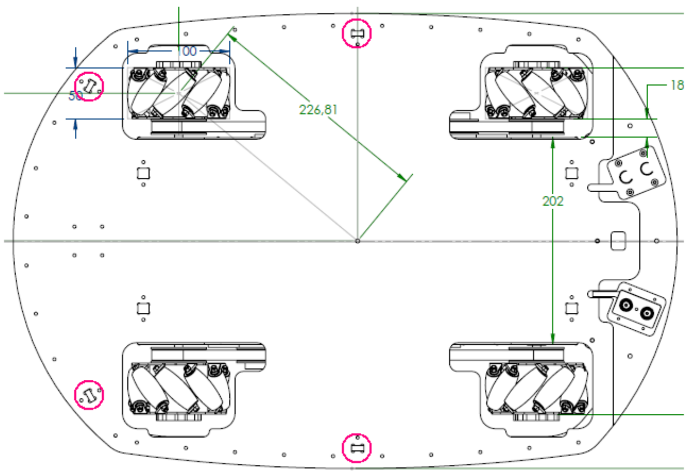

Figure 58: positions of the IR ground sensors

5.4. IMU Board

The IMU Board, depicted in Fig. 59, uses the MPU6050 that combines a MEMS 3-axis gyroscope and a 3-axis accelerometer on the same silicon die together with an onboard Digital Motion Processor™ (DMP™) capable of processing complex 9-axis MotionFusion algorithms. This chip communicates with a microcontroller using I2C communication. This microcontroller will be able to communicate with the navigation computer by emulating a USB to COM serial port.

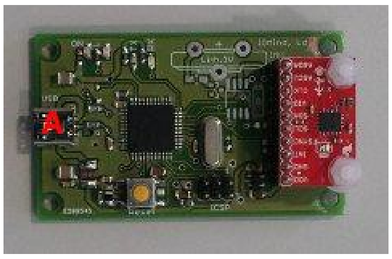

Figure 59: IMU Board

5.5. Interaction Board

The Interaction Board is depicted in Fig. 60. This board allows the PC-NAV to control the arms and the head of the robot, the body LED lights, the mouth LED dot matrix, the enable/disable of the projector and the body capacitive sensors. The Master Interaction Controller uses a **PIC18F6527** microcontroller to control and manage all the communication between the high-level robot PC-NAV and the interaction devices. A Slave PI Motor Controller uses a **PIC18F2431** microcontroller to control the robot's head pan motor.

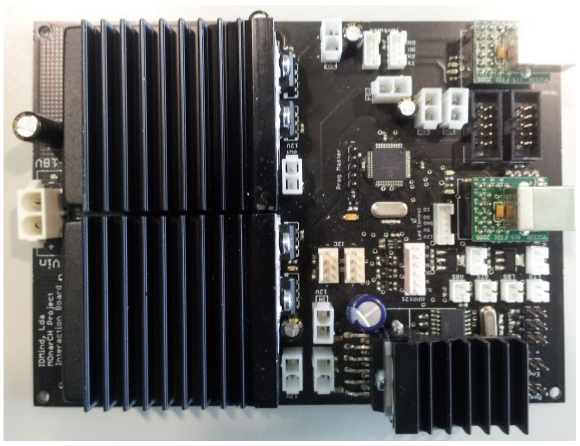

Figure 60: MOnarCH robot Interaction Board

The overall architecture of the Interaction Board is depicted in Fig. 61.

The interaction board communicates with the PC-NAV using two USB-to-RS232 converters. One of the converters will connect directly to the Herkulex arm motors, no processing is needed by the board. The board will supply a 9V power supply to power these motors. The other converter will connect to the Master Interaction Controller.

The Master Interaction Controller will:

- provide the needed power supply voltages for the different interaction devices;
- receive commands from the PC-NAV and send back sensor Data;
- implement the needed communication (I2C, SPI and Serial) to control the different interaction devices;
- control the fade in and fade out for the RGB LEDs;
- read the information from the capacitive sensors;
- read the arms' limit switches.

The Slaves PI Motor Controllers will:

- receive velocity and position references from the Master Interaction Controller;
- periodically read the encoder pulses and potentiometer position;
- send the estimated position of the head and send it to the Master Interaction Controller.

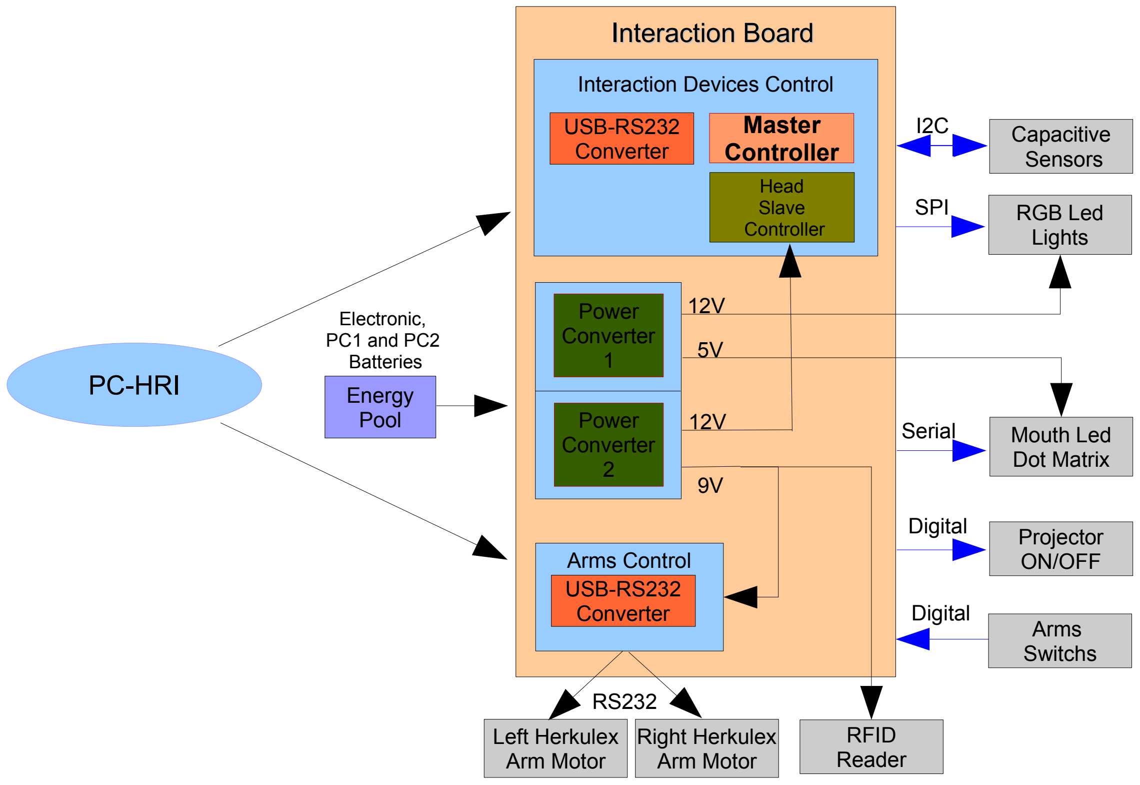

Figure 61: MOnarCH Interaction Board Architecture

6. Charger Docking Station

One important capability of the robot is the possibility to work without human intervention. To achieve this point the robot must be able to manage the onboard power and autonomously charge itself. One charger docking station was developed to be installed in a service area, where the robot can enter and plug itself in. This docking station will provide the necessary power that the onboard battery charger needs to charge the batteries. The Docking Station is a passive power station: control of the charging process is managed by the onboard Sensor&Management Board. The Docking Station will be equipped with a 320W[1] switch mode power supply able to power all the onboard equipment and at the same time provide about 2,5A of current for each of the 4 battery chargers. Fig. 62 (left image) presents the docking station and the 320W power supply. Fig. 62 (right image) shows the robot charging mechanism that will attach to the docking station. Fig. 63 depicts the MOnarCH robot during the charging process.

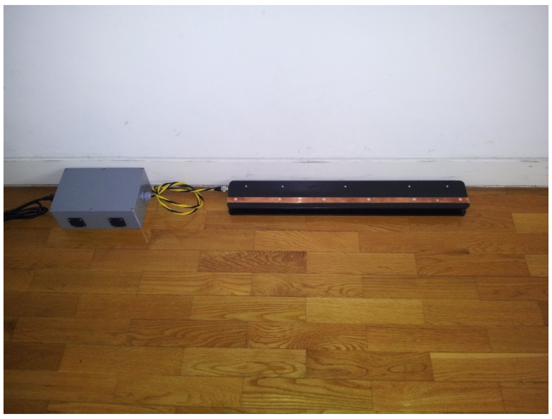
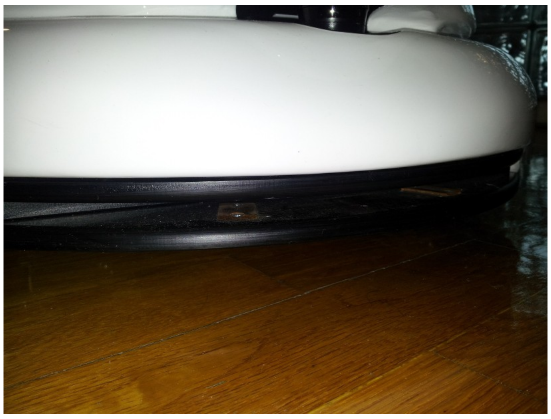

Figure 62: MOnarCH Docking Station (left image) and robot docking mechanism

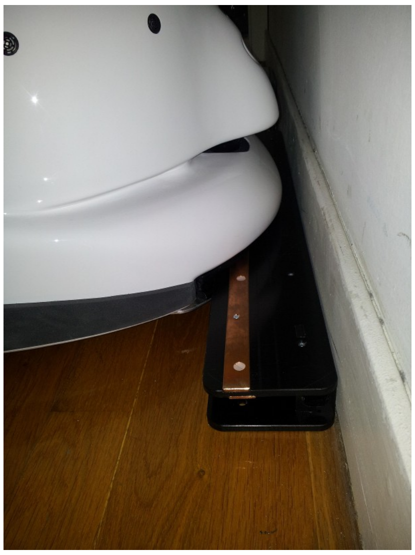
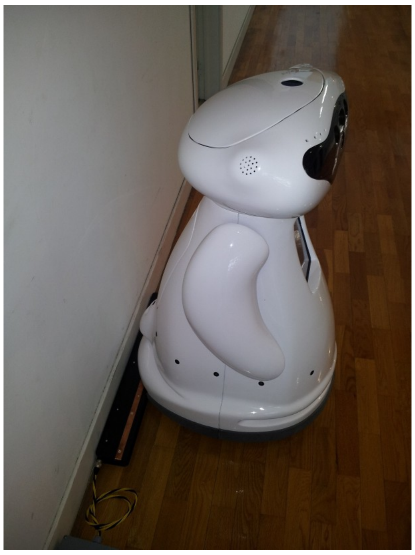

Figure 63: MonarCH robot charging at the docking station

1 The power supply can be adjusted to charge multiple robots.

7. Working with the Robot

The following subsections explain the steps to set up the robot for normal operation.

7.1. Setting Up the Hardware

To be able to control the MOnarCH robot the user must perform the following steps:

1. Remove the rear button cover from the back of the robot (Fig. 64).

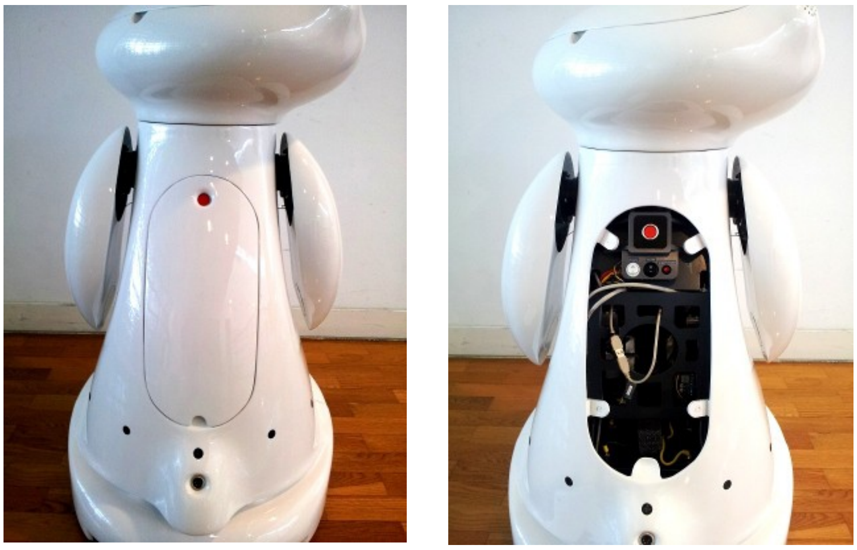

Figure 64: MOnarCH rear cover access

2. Press the Electronics' **Robot Power** button. By turning on the Electronics power button all the auxiliary batteries will be enabled, providing power to all onboard components. This means that the internal power sources of the PC-NAV and PC-HRI computers are powered (Fig. 65).

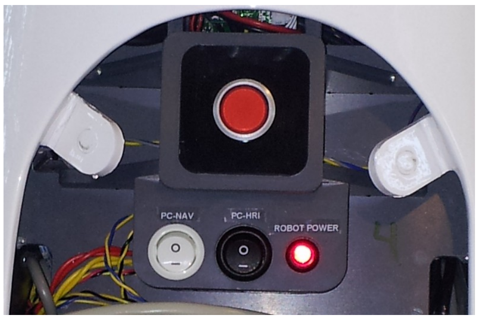

Figure 65: Robot Power button

3. If the **PC-NAV** power button is OFF then the user must turn it ON to power this computer. If the switch was already ON the computer will boot once the **Robot Power** button is powered on (Fig. 66)

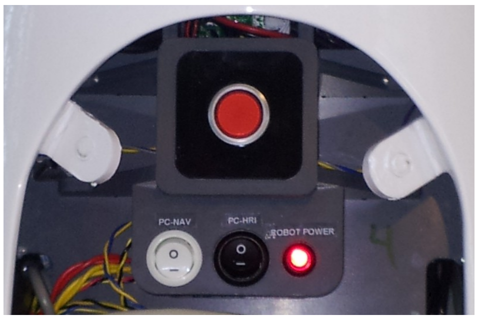

Figure 66 : PC-NAV button ON

4. (For the SO robot only) If the **PC-HRI** power button is OFF then the user must turn it ON to power this computer. If the switch was already ON the computer will boot once the **Robot Power** button is powered on (Fig. 67).

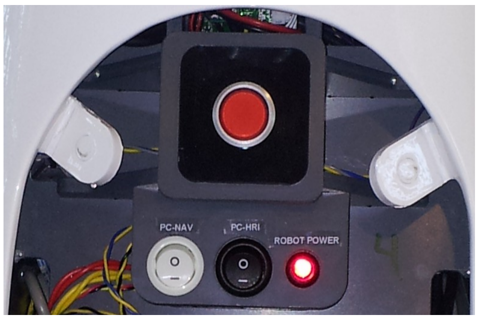

Figure 67 : PC-HRI button ON

5. To use the motors, press the **Motor/E-Stop** button (Fig. 68).

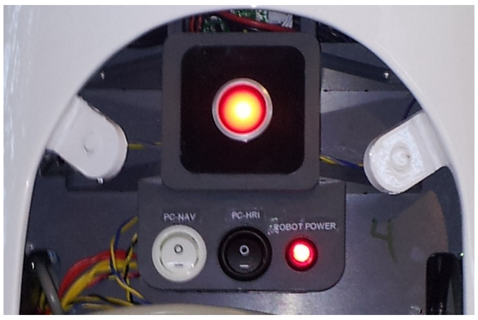

Figure 68 : Motor/E-Stop Button ON

6. Finally, put back the rear cover on the robot. The motor power button will be available, allowing the user to quickly enable or disable the power of the motors, and also, put and remove the robot from charge as explained in section 7.3 (Fig. 69).

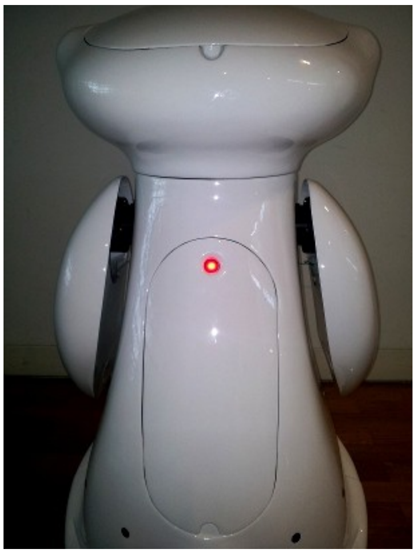
Figure 69 : Motor button ON

At this moment the robot platform can be controlled by the PC-NAV.

7.2. Shutting Down the MOnarCH Robot

To shut down the MOnarCH robot the first thing to do is to disable the motors by removing the power from the motor drivers. To do that the user should:

1. push the **Motor/E-Stop** button on the back of the robot;
2. If the computers are switch ON, then the user should send a shut down command to all the computers and wait until they are all OFF or switch OFF the computers by turning off the **PC-NAV** and **PC-HRI** buttons OFF.
3. When the fans of both computers stop moving, then the user can disconnect the **Robot Power** button.

7.3. Charging the Batteries

The quick way to charge the batteries is to place the robot in the charging Docking Station or to plug the charger connector directly to robot power plug on the back. The Docking Station/Charger should be powered and all the batteries and power buttons (Robot Power and Motors) should be ON.

The Docking Station/charger will power the electronics and all the computers.

If the **Motor/E-Stop** button on the back of the robot is ON the robot will start to charge all the batteries. If the **Motor/E-Stop** button is OFF, then the user should press the **Motor/E-Stop** button to its ON position. Then the robot will start to charge all the batteries.

The charging procedure is fully automatic. When the batteries are fully charged the charger will stop the charge. To prevent any damage it was introduced a charging limit of 14.6V for each battery. If the charge voltage reaches this value the Sensor&Management board will remove the battery from charge.

To return from the charge mode two approaches can be used:

- Switch OFF the **Motor/E-Stop** button and remove the robot from the charger by hand. By using this procedure the Sensor&Management Board will disconnect all the batteries from the charger and put them all in enable mode.
- Use the PC-NAV to switch back all the batteries from charge mode to enable/disable mode. With this procedure the user can avoid current peaks that can reset the computers and also allow the platform to autonomously enter and leave the Docking Station.

8. Mbot_ROS

8.1. User Tasks

8.1.1. Dependencies

mbot_ros uses custom-defined ROS messages that belong to the monarch_msgs package. In order to be able to use ROS tools to manipulate these custom message types, the monarch_msgs package should be available in your ROS system (for instance, you should be able to roscd into the monarch_msgs package).

8.1.2. Installation

This section describes the installation of software provided by SELFTECH that makes all of the robots' hardware available for usage within a ROS environment.

Obtaining installation package

The mbot-ros software is distributed as a ready-to-install debian package. You may download it from `https://selftech.com/monarch/`. Username is `monarch` and password is `hominibus`.

Installing software

Once you download the package you may install it with the command `dpkg -i mbot_ros-<version number>-Linux.deb`.

Uninstalling is just as simple with the command `dpkg -r mbot_ros`.

Edit namespace in launch file in order to match the robot name (e.g. “`mbot04`”): “`rosed mbot_ros mbot.launch`”

Finally, add the following to all users’ `.bashrc` file:

- source /opt/ros/hydro/setup.bash
- source /opt/monarch_msgs/setup.bash
- source /opt/mbot_ros/setup.bash
- export ROS_MASTER_URI=http://mbotXX:11311 (where XX is the mbot number, 01, 02, 03, etc...)
- export MBOT_NAME=mbotXX (where XX is the mbot number, 01, 02, 03, etc...)

After doing this change either log out and log back in again or run in every terminal `source ~/.bashrc`.

Since mbot_ros installs `udev` configuration files, `udev` should be restarted prior to using the software. Either reboot the system or issue the command `sudo service udev restart`.

```
 <frequency>45</frequency>

  </idmind_imu>
```

```
    <idmind_motor_board>
      <enabled>true</enabled>
      <device_path>/dev/mbot-motorboard</device_path>
      <clicks_per_turn>2000</clicks_per_turn>
      <gear_ratio>13.795918367</gear_ratio>
      <hardstop_time>2</hardstop_time>
      <frequency>45</frequency>
      <vel_timeout_ms>200</vel_timeout_ms>
    </idmind_motor_board>
    <idmind_sensor_board>
      <enabled>true</enabled>
      <device_path>/dev/mbot-sensorboard</device_path>
      <frequency>10</frequency>
      <num_sonars>12</num_sonars>
      <sonars_fov>0.785398163</sonars_fov>
      <sonars_max_range>6.4516</sonars_max_range>
      <sonars_min_range>0.15240</sonars_min_range>
    </idmind_sensor_board>
    <joystick>
      <enabled>true</enabled>
      <device_path>/dev/input/js0</device_path>
      <angular_gain>3.14</angular_gain>
      <linear_gain>1.0</linear_gain>
    </joystick>
  </devices>
</mbot_ros>
```

Item path	Description	Values	Default Value	Added in version
`config_version`	Indicates the version number of the configuration file structure	Integer greater than 0	`0`	1
`autostart`	Section that contains configurations for mbot-ros automated startup			1

`autostart / enabled`	Controls whether mbotrosd will run at startup	`true` or `false`	`false`	1
`platform`	Section that defines which physical platform is the software running on			2
`platform / type`	Type of the platform	`mbot_po` or `mbot_so` depending on whether the platform is a PO or SO robot.	`mbot_so`	2
`four_wheel_mecanum`	Four wheel mecanum kinematic configurations.			2
`four_wheel_mecanum / l1`	L1	double	0.14429	2
`four_wheel_mecanum / l2`	L2	double	0.175	2
`four_wheel_mecanum / r`	Wheel radius	double	0.05	2
`devices`	Section that contains configurations for the devices that mbotrosd interfaces directly			2
`devices / idmind_imu`	Configuration for IdMind's IMU board			2
`devices / idmind_imu / enabled`	Enables mbot_ros to connect to this board	`true` or `false`	`true`	2
`devices / idmind_imu / device_path`	Filesystem path to the device	string	`/dev/mbot-imu`	2
`devices / idmind_imu / frequency`	Frequency at which data is published, in Hz	double	45	2
`devices / idmind_motor_board`	Configuration for IdMind's motor board			2
`devices / idmind_motor_board`	Enables mbot_ros to connect to this board	`true` or `false`	`true`	2

`/ enabled`				
`devices / idmind_motor_board / device_path`	Filesystem path to the device	string	`/dev/mbot-motorboard`	2
`devices / idmind_motor_board / clicks_per_turn`	Number of encoder clicks per motor rotation	integer	2000	2
`devices / idmind_motor_board / gear_ratio`	Gear ratio, from motor to wheels (number of rotation of the motor per wheel rotation)	double	13.795918367	2
`devices / idmind_motor_board / hardstop_time`	Time duration, in seconds, while a hardware hardstop will be active in case of the bumpers being activated	integer	2	2
`devices / idmind_motor_board / frequency`	Frequency at which data is published, in Hz	double	45	2
`devices / idmind_motor_board / vel_timeout_ms`	Maximum time to continue sending the same speed value to the electronics, in ms	unsigned int	200	2
`devices / idmind_sensor_board`	Configuration for IdMind's sensor board			2
`devices / idmind_sensor_board / enabled`	Enables mbot_ros to connect to this board	`true` or `false`	`true`	2
`devices / idmind_sensor_board / device_path`	Filesystem path to the device	string	`/dev/mbot-sensorboard`	2
`devices / idmind_sensor_board / frequency`	Frequency at which data is published, in Hz	double	10	2
`devices / idmind_sensor_board / num_sonars`	Number of sonars installed on the robot	integer	12	2
`devices / idmind_sensor_board / num_sonars`	Number of sonars installed on the robot	integer	12	2

`devices / idmind_sensor_board / shutdown_voltage`	Voltage under which the PC is automatically shutdown by mbot_ros in order to prevent a hard shutdown	11.5	2	
`devices / idmind_sensor_board / sonars_fov`	Field-of-view of the sonars, in radians	integer	0.785398163	2
`devices / idmind_sensor_board / sonars_max_range`	Maximum range detected by sonars	double	6.4516	2
`devices / idmind_sensor_board / sonars_min_range`	Minimum range detected by sonars	double	0.1524	2
`devices / joystick`	Configuration for joystick capabilities	2		`devices / joystick`
`devices / joystick / enabled`	Enables mbot_ros to connect to the joystick			2
`devices / joystick / enabled / device_path`	Filesystem path to the device	string	`/ dev/input/js0`	2
`devices / joystick / enabled / angular_gain`	Gain of the angular joystick commands. Value will equal the maximum angular speed in rad/s	double	3.14	2
`devices / joystick / enabled / linear_gain`	Gain of the linear joystick commands. Value will equal the maximum angular speed in rad/s	double	1.0	2

Table 3: mbot-ros configuration XML description.

8.1.3. Controlling startup and shutdown of mbot_ros

The software package mbot_ros contains a daemon named mbotrosd which is able to execute mbot_ros and `roscore` at system startup. This daemon is configured by default to start automatically at system startup. The daemon's status can be checked with the command `sudo service mbotrosd status`. Likewise it can be manually started using `sudo service mbotrosd start` and manually stopped using `sudo service mbotrosd stop`. As long as `mbotrosd` is running, all sensors and actuators of the robot should be available from ROS topics.

Since a ROS master is needed to have the software running, `mbotrosd` checks if a ROS master is available and, if not, will start a `roscore` instance itself.

If the user wants to prevent mbotrosd from starting automatically at startup it is just a matter of setting the `autostart / enabled` field in the configuration file to “`false`”.

If mbotrosd autostart is disabled, it is still possible to run mbot_ros with the command `rosrun mbot_ros mbotros`. Please note that a `roscore` instance should be available prior to running mbot_ros.

8.1.4. Automatic system shutdown

In order to prevent the system from suffering a hard shutdown, an automatic shutdown feature was added. Once the PC voltage goes below a pre-defined limit (defined in the `config.xml` file in section `devices / idmind_sensor_board / shutdown_voltage`), mbot_ros issues a `shutdown` command. A 2 minute notice is given to all logged in users with the following message appearing on all open consoles:

The system is going down for power off in 2 minutes!

PC voltage too low!

8.1.5. View log files

The mbotrosd daemon logs to the standard system log file `/var/log/syslog`. An example of log entries produced by `mbotrosd` follows:

```
Nov 11 12:09:42 st-desk01 mbotrosd[4237]: starting

Nov 11 12:09:42 st-desk01 mbotrosd[4237]: daemonization complete without effort

Nov 11 12:09:42 st-desk01 mbotrosd[4238]: startup finished

Nov 11 12:09:42 st-desk01 mbotrosd[4238]: loading configuration

Nov 11 12:09:42 st-desk01 mbotrosd[4238]: configuration loaded

Nov 11 12:09:42 st-desk01 mbotrosd[4238]: time has passed

Nov 11 12:09:46  mbotrosd[4238]: last message repeated 4 times

Nov 11 12:09:46 st-desk01 mbotrosd[4238]: received signal 15 (Terminated)

Nov 11 12:09:46 st-desk01 mbotrosd[4238]: terminated
```

A user may view this information in real time using the command `tail -f /var/log/syslog | grep mbotros`. This will show only messages originating from the mbotrosd daemon.

8.1.6. Visualizing robot information on rviz

Be sure that your computer is in the same network as the robot and set the environment variable `ROS_MASTER_URI` to point to the ROS master instance running onboard the robot, for instance: `export ROS_MASTER_URI=http://mbot01:11311/`.

Then run `rviz` using `rosrun rviz rviz`. Once `rviz` is running, add visualizations for the available topics to see whatever you need.

A ready-to-use configuration for `rviz` is available in the mbot_ros directory with the name `mbot.rviz`.

8.1.7. Driving the robot manually

mbotrosd automatically uses USB joysticks that are connected to the robot. Once a joystick is connected, `mbotrosd` will publish target velocities to the `/cmd_vel_manual` topic and this will make the robot move according to the joystick input.

Joystick functionality was tested with a Logitech RumblePad 2 joystick that has keys as indicated in Fig. 70.

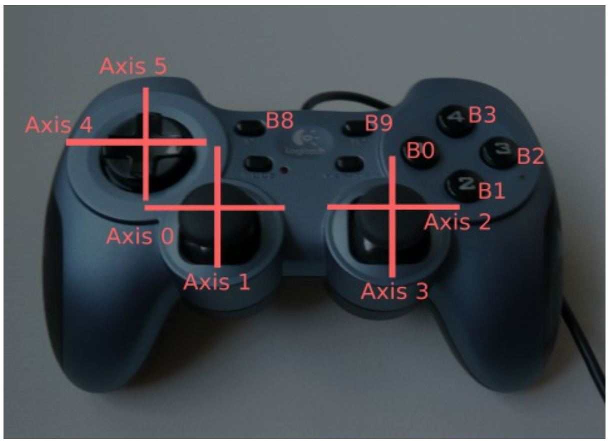

Fig. 70: Joystick used for testing.

The key mappings of the joystick are defined in Table 4.

Key	Mapped to
Axis 0	Robot Linear Y-axis speed
Axis 1	Robot Linear X-axis speed
Axis 2	Robot Angular speed
Axis 4	Increase variable gain multiplier
Axis 5	Decrease variable gain multiplier

Table 4: Joystick key mappings.

Please note that if other software is also publishing data to the `/cmd_vel` topic, the robot will only follow the `/cmd_vel_manual`, thus manual control overrides automatic control.

8.2. Reference documentation

8.2.1. Robot reference frames

ROS provides a framework for registering relationships between coordinate frames over time named `tf`. Each of the robots' sensors measure features in its own coordinate frame. In order to successfully be able to relate information gathered from different sensors, the frame transformation data must be regularly published. This is done using a `static_transform_publisher`.

The base frame of all sensor frames is named "`base_link`". For the mbots this frame is defined as being in the 2D geometrical center of the robots' wheels, at the ground level. Therefore, if the robot is on a perfectly horizontal plane, the point (0,0,0) in the "`base_link`" is the point on the floor that is at the same distance from all of the wheels' centers.

All sensor frames follow the naming pattern "`<sensor_name><sensor_number>`" . Table 5 lists all the publicized frames and their posture relative to the "`base_link`" frame.

Besides the sensors positions, the robot's odometry position is published in `tf` as well, with the `base_link`'s posture being published within an `odom` frame.

Frame name	**Description**	**Translation (x,y,z)**	**Rotation (pitch, roll, yaw)**
`imu01`	First inertial measurement unit.	(0, 0, 0)	(0, 0, 0)
`lrf01`	First laser range findes.	(0.3, 0, 0.1365)	(0, 0, 0)
`rgbd01`	First RGB-D camera.	(0.3, 0, 1.0)	(0, 0, 0)
`rgbd02`	Second RGB-D camera.	(0, 0, 0)	(0, 0, 0)
`sonarN`, n=01…12	Nth sonar.	($0.15\cos(angle)$, $0.15\sin(angle)$,0)	(0, 0, angle), angle = $\frac{2\pi(N-1)}{12}$

Table 5: Sensor frames.

8.2.2. Published topics

Data collected from onboard sensors is published in ROS topics. An effort was made to use as much as possible standard message types and topic names in order to make it easier to use existing ROS software with the mbots. Table 6 lists the publicized topics and which sensor data they contain. Some topics have custom data types that belong to the `monarch_msgs` package.

Topic name	**Data type**	**Description**
`auxiliary_batteries_voltage`	`AuxiliaryBatteriesVoltage`	Readings from auxiliary batteries voltage sensors.

batteries_voltage	BatteriesVoltage	Readings from main batteries voltage sensors.
bumpers	BumpersReadings	Readings obtained from the robots bumpers.
charger_status	ChargerStatus	Readings from charger lines voltage sensors.
ground_sensors	GroundSensorsReadings	Readings from ground sensors.
hardstop_status	HardstopStatus	Status of the low level hardstop.
hokuyo_node/parameter_descriptions	TBD	TBD
hokuyo_node/parameter_updates	TBD	TBD
joy	sensor_msgs::Joy	Readings from USB Joystick.
motor_board_communication_status	MotorBoardCommunicationStatusReadings	Not yet implemented.
motor_board_cooling_fans	MotorsCoolingFans	Motor board cooling fans status.
motor_board_temperatures	MotorBoardTemperatures	Motor board temperature readings.
motor_board_voltages	MotorBoardVoltages	Motor board voltage readings.
odom	nav_msgs::Odometry	Odometry measurements calculated from measured wheel rotation. Note: the robot's posture (the `base_link` frame) is also published using `tf` within the `odom` frame.
relativeHumidity	sensor_msgs::RelativeHumidity	Range measurements collected from the onboard sonars.
sensor_board_communication_status	SensorBoardCommunicationStatusReadings	Not yet implemented.
sonars	sensor_msgs::Range	Range measurements collected from the onboard sonars.
temperature	sensor_msgs::Temperature	Ambient temperature measured by the robot.

Table 6: Published sensor topics.

8.2.3. Subscribed topics

mbot_ros expects to receive information from other software in a set of specific topics. These are mainly related to control signals to be sent to the robot's actuators. Table 7 lists the expected topics and what is done with the information they contain.

Topic name	Data type	Description
`cmd_vel`	`geometry_msgs::Twist`	Target speeds of the robot in the robot frame ("`base_link`"), published by an automatic control source.
`cmd_vel_manual`	`geometry_msgs::Twist`	Target speeds of the robot in the robot frame ("`base_link`"), published by a manual control source.
`set_state_imu`	`SetStateImu`	Set the Inertial Measurement Unit state.
`set_motor_board_cooling_fans`	`SetStateMotorBoardCoolingFans`	Allow automatic or manual control of the fans to cool the motors and drivers.
`set_state_aux_batt1_power`	`SetStateAuxiliaryPowerBattery`	Enable/Disable/Charge auxiliary power battery 1.
`set_state_aux_batt2_power`	`SetStateAuxiliaryPowerBattery`	Enable/Disable/Charge auxiliary power battery 2.
`set_state_electronics_power`	`SetStateElectronicPower`	Enable/Charge electronic power battery.
`set_state_motors_power`	`SetStateMotorsPower`	Enable/Disable/Charge motors power battery.
`set_state_sonars`	`SetStateSonars`	Enable/Disable sonars.

Table 7: Subscribed topics

8.2.4. mbot Kinematics Transformations

The mbot's kinematic model was adapted from Doroftei's et al paper titled "Omnidirectional Mobile Robot--Design and Implementation". Adjustments were made in order to account for different wheel rotation direction and wheel numbering.

Considering that:

- l_1 and l_2 are the *x*-axis and *y*-axis distances of the wheel center to the robot frame's origin.
- *r* is the wheel radius.

- the robot’s wheels are number from 1 to 4 in counter-clockwise direction starting from the wheel that is in the robot’s front left corner.
- ω_k represents the angular speed of wheel *k*
- *vx*, *vy* and □ represent the robot’s instantaneous speeds

$$L = l_1 + l_2 \quad (2.1)$$

Inverse Kinematics

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = \frac{1}{r} \begin{bmatrix} -1 & -1 & L \\ -1 & +1 & L \\ +1 & +1 & L \\ +1 & -1 & L \end{bmatrix} \cdot \begin{bmatrix} vx \\ vy \\ \dot{\theta} \end{bmatrix} \quad (2.2)$$

Forward Kinematics

$$\begin{bmatrix} vx \\ vy \\ \dot{\theta} \end{bmatrix} = \frac{r}{4} \begin{bmatrix} -1 & -1 & +1 & +1 \\ -1 & +1 & +1 & -1 \\ \frac{1}{L} & \frac{1}{L} & \frac{1}{L} & \frac{1}{L} \end{bmatrix} \cdot \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} \quad (2.3)$$

8.3. Experimental Results

8.3.1. Hardware Communication Latencies

In order to assess the appropriate update rates for data acquired from the mbots hardware, we conducted an experiment in which we measured the latency of each of three types of *interactions* with the robots’ hardware. We consider an *interaction* to be the actions needed to acquire (or set) an individual piece of data. In this study, we focused on the three interactions that have higher update rates:

GetEncoders

Get encoder information in order to calculate odometry.

SetVelocity

Set wheel velocities.

GetImu

Get IMU information.

We consider that the interaction starts in the instant the software running on the main computer is ready to send data to the hardware and it stops when the complete response has been received and correctly parsed by the software. The experiment duration was of roughly 1 hour.

The results obtained are showed in Fig. 71 and statistics on this data is showed in Table 8. The `GetEncoders` and `SetVelocity` interactions belong to the same hardware board, so they must not occur simultaneously. By adding their 99.9% percentile time we obtain a total time of $0.0202150s$, which would result in a maximum frequency of about $49.46Hz$. As for the IMU, with a value of $0.0209570s$ this would result in a maximum frequency of about $47.7Hz$. In order to give a bit of slack to have a "round" number, it was decided that these devices should work at a frequency of $45Hz$.

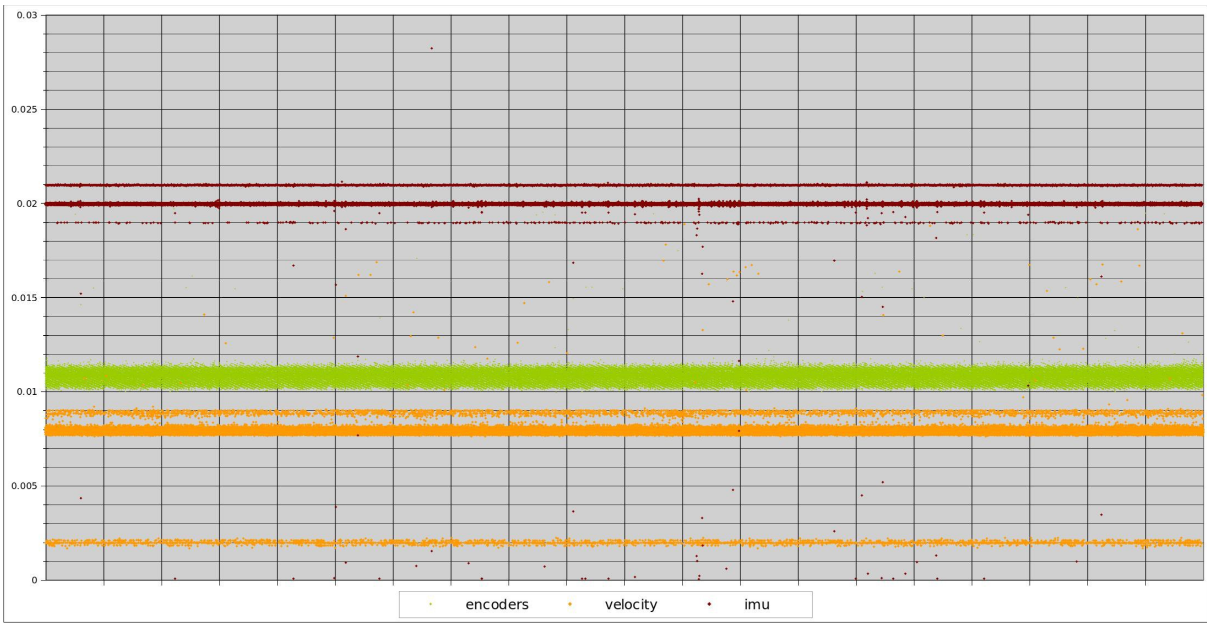

Figure 71: Interactions latencies per data type.

	`GetEncoders`	`SetVelocity`	`GetImu`
min	0.0099510	0.0016790	0.0000400
max	0.0196990	0.0188820	0.0282170
avg	0.0108120	0.0078534	0.0199712
stddev	0.0002853	0.0007979	0.0003140
median	0.0108410	0.0079650	0.0199600
99.9% percentile	0.0113580	0.0088570	0.0209570

Table 8: Interactions latency statistics per data type.

Annex A. Board Controllers Software Protocol

The Onboard Navigation Computer is able to communicate with the Motors and Sensor&Management boards using two independent USB Ports. In each board a FTDI USB to RS232 converter converts the USB data connection to a TTL Serial PORT data connection for the microcontrollers on the Boards.

All the USB board connectors can be connected using the following UART settings:

Baud rate: 115200bps

Data bits: 8

Stop bits: 1

Parity bit: No parity

HW Flow Control: Disable

To communicate with the robot boards the following protocol is used:

Write -> [Command][First Byte][Second Byte][....]

Read -> [Command][First Byte][Second Byte] [....][Send Number][CheckSum_H][CheckSum_L]

Where:

[Command] is the set or get command;

[Send Number] is an incremental number that counts all the sends that each microcontroller has already performed;

[CheckSum_H:CheckSum_L] is the checksum of the [Command]+[First Byte]+[Second Byte]+ [....]+[Send Number].

A.1. Sensor&Management Communication Protocol (board1)

Enable/Disable/Charge Motors Power:

[Command] = 0x45

Write -> [0x45][Control]

Read -> [0x45] [Send Number] [CheckSum_H] [CheckSum_L]

where:

[Control] = 0 : Motor power Disabled;

[Control] =1 : Charge motor power enable Mode;

[Control] = 2 : Motor power Enable.

Enable/Disable/Charge Auxiliary Power battery 1 to 2:

[Command] = 0x40 to 0x41

Write -> [0x40 to 0x41][Control]

Read -> [0x40 to 0x41] [Send Number] [CheckSum_H] [CheckSum_L]

where:

[Control] = 0 : Auxiliary power battery (1 to 2) Disabled;

[Control] = 1 : Charge auxiliary power battery (1 to 2) enable Mode;

[Control] = 2 : Auxiliary power battery (1 to 2) Enable.

Enable/Charge Electronic Power battery:

[Command] = 0x46

Write -> [0x46][Control]

Read -> [0x46] [Send Number] [CheckSum_H] [CheckSum_L]

where:

[Control] =1 : Charge electronic battery enable Mode;

[Control] =2 : Electronic power battery Enable.

Get Batteries Voltage command:

[Command] = 0x51

Write -> [0x51]

Read -> [0x51] [Motor Battery] [Electronics Battery] [Charger] [Send Number] [CheckSum_H] [CheckSum_L]

Where:

[Motor Battery] is the Motor battery voltage. When in charging mode the value is +/- 0V.

To obtain the real measured voltage of this battery the following formula must be used:

Motor battery voltage = (double)((double)[Motor Battery] / 10.0);

[Electronics Battery] is the battery that powers all the electronics and the onboard Navigation Computer.

To obtain the real measured voltage of this battery the following formula must be used:

Electronics battery voltage = (double)((double)[Electronics Battery] / 10.0);

[Charger] is the power from the docking station.

To obtain the real measured voltage of this battery the following formula must be used:

Charger voltage = (double)((double)[Charger] / 10.0);

Get Auxiliary Batteries Voltage command:

[Command] = 0x52

Write -> [0x52]

Read -> [0x52] [PC1 Battery] [PC2 Battery] [Send Number] [CheckSum_H] [CheckSum_L]

Where:

[PC1-PC2 Battery] are auxiliary batteries that can be used to power computers or other electronic devices that the robot can carry.

To obtain the real measured voltage of this battery the following formula must be used:

PC1-PC2 battery voltage = (double)((double)[PC1-PC2] / 10.0);

Get Bumpers command:

[Command] = 0x53

Write -> [0x53]

Read -> [0x53] [Bumpers][Send Number] [CheckSum_H] [CheckSum_L]

Where:

[Bumpers] allows the detection of collisions in different parts of the robot. The received [bumpers] byte indicates if a bump is being detected. Bumper byte has the following composition [0000ABCD] where:

[A] is the right bump;

[B] is the left bump;

[C] is the front bump;

[D] is the rear bump.

Each bit can assume two possible values: 0 if no bump is being detected or 1 if a bump is being detected.

Get Sonars command:

[Command] = 0x54

Write -> [0x54]

Read -> [0x54] [Sonar0_High_Byte] [Sonar0_High_Byte] ---- [Sonar11_High_Byte] [Sonar11_High_Byte] [Send Number][CheckSum_H] [CheckSum_L]

The get sonars command will retrieve the distance information of 12 sonar sensor in cm. To get the distance value in cm the following calculation must be performed:

Sonar 0..11 = (int)(0,013895*([Sonar0..11_High_Byte]*256+[Sonar0..11_Low_Byte]));

Enable/Disable Sonars command:

[Command] = 0x55

Write -> [0x55][Control]

Read -> [0x55] [Send Number] [CheckSum_H] [CheckSum_L]

where:

[Control] = 0 : Disable sonar readings;

[Control] = 1 : Enable sonar readings.

Get Environment sensors command:

[Command] = 0x56

Write -> [0x56]

Read -> [0x56] [Temperature] [Humidity] [Send Number][CheckSum_H] [CheckSum_L]

The Environment sensor command retrieves the information of the environment temperature and humidity where:

[Temperature] retrieves a temperature between 0ºC to 125ºC;

[Humidity] retrieves the relative humidity between 0% and 100%;

Get Charger Status command:

[Command] = 0x58

Write -> [0x58]

Read -> [0x58] [Charger_Status][Send Number][CheckSum_H] [CheckSum_L]

where:

[Charger Status] retrieves the information of the Charger Status. [XXXXABCD]retrieves the information of the Charger Status. [XXXXABCD]

[A]=0 :PC2 Battery Not Charging

[A]=1 :PC2 Battery Charging

[B]=0 :PC1 Battery Not Charging

[B]=1 :PC1 Battery Charging

[C]=0 : Motor Battery Not Charging

[C]=1 : Motor Battery Charging

[D]=0 : Electronic Battery Not Charging

[D]=1 : Electronic Battery Charging

This information is only valid if the robot is connected to the Charger with a voltage higher that 16V.

Get Ground Sensors Values:

[Command]=0x59

Write ->[0x59]

Read ->[0x59] [Right_Sensor][Right_Front_Sensor][Left_Front_Sensor][Left_Sensor][Send Number][CheckSum_H] [CheckSum_L]

Get Firmware version number command:

[Command] = 0x20

Write -> [0x20]

Read -> [0x20] [Firmware information (25bytes)] [Send Number] [CheckSum_H] [CheckSum_L]

where:

[Firmware information] is a set of 25 bytes in ascii with the following information:

“Board1 fw A.BC YYYY/MM/DD” for example “Board1 fw 1.00 2014/02/14”

A is the version and BC the revision number.

A.2. Motor Board Communication Protocol (board2)

Set Velocity command:

[Command] = 0x56

[Left front wheel Velocity]=[Left front wheel Velocity High][Left front wheel Velocity Low]

[Right front wheel Velocity]=[Right front wheel Velocity High][Right front wheel Velocity Low]

[Left rear wheel Velocity]=[Left rear wheel Velocity High][Left rear wheel Velocity Low]

[Right rear wheel Velocity]=[Right rear wheel Velocity High][Right rear wheel Velocity Low]

Write -> [0x56][Left front wheel Velocity High][Left front wheel Velocity Low][Right front wheel Velocity High][Right front wheel Velocity Low][Left rear wheel Velocity High][Left rear wheel Velocity Low][Right rear wheel Velocity High][Right rear wheel Velocity Low]

Read -> [0x56][Send Number][CheckSum_H][CheckSum_L]

To obtain the High and Low bytes use:

Left front wheel Velocity High = (byte) (Left front wheel Velocity >> 8);

Left front wheel Velocity Low = (byte) (Left front wheel Velocity & 0xFF);

Right front wheel Velocity High = (byte) (Right front wheel Velocity >> 8);

Right front wheel Velocity Low = (byte) (Right front wheel Velocity & 0xFF);

Left rear wheel Velocity High = (byte) (Left rear wheel Velocity >> 8);

Left rear wheel Velocity Low = (byte) (Left rear wheel Velocity & 0xFF);

Right rear wheel Velocity High = (byte) (Right rear wheel Velocity >> 8);

Right rear wheel Velocity Low = (byte) (Right rear wheel Velocity & 0xFF);

Get Encoders Ticks command:

[Command] = 0x4A

Write -> [0x4A]

Read -> [0x4A][Left front Motor Ticks High][Left front Motor Ticks Low][Right front Motor Ticks High][Right front Motor Ticks Low][Left rear Motor Ticks High][Left rear Motor Ticks Low][Right rear Motor Ticks High][Right rear Motor Ticks Low][Send Number][CheckSum_H][CheckSum_L]

To get the Left front Motor Ticks:

Left front Motor Ticks =(int)Left front Motor Ticks High*256+Left front Motor Ticks Low;

To get the Right front Motor Ticks:

Right front Motor Ticks =(int)Right front Motor Ticks High*256+Right front Motor Ticks Low;

To get the Left rear Motor Ticks:

Left rear Motor Ticks =(int)Left rear Motor Ticks High*256+Left rear Motor Ticks Low;

To get the Right rear Motor Ticks:

Right rear Motor Ticks =(int)Right rear Motor Ticks High*256+Right rear Motor Ticks Low;

To calculate the CheckSum use the following formula:

CheckSum = 0x4A + Left front Motor Ticks High + Left front Motor Ticks Low +
Right front Motor Ticks High + Right front Motor Ticks Low +
Left rear Motor Ticks High + Left rear Motor Ticks Low +
Right rear Motor Ticks High + Right rear Motor Ticks Low + Send Number;

Get Motor Board Voltage command:

[Command] = 0x51

Write -> [0x51]

Read -> [0x51] [Motor Voltage] [Driver Voltage] [Electronics Voltage] [Power Status] [Send Number] [CheckSum_H] [CheckSum_L]

Where:

[Motor Voltage] is the voltage measure of the power supply of the motors. In normal operation will have a value between 9.5V and 13V.

Motor voltage power = (double)((double)[Motor Voltage] / 10.0);

[Driver Voltage] is the voltage measured of the power supply of the drivers for the motors. In normal operation the value should be between 12 to 15V.

Driver voltage power = (double)((double)[Driver Voltage] / 10.0);

[Electronics Voltage] is the voltage measured of the power supply of the control electronics. In normal operation the value should be between 9.5 to 13V.

Electronics voltage power = (double)((double)[Electronics Voltage] / 10.0);

[Power Status] is the voltage status [AXXXXBCD].

where:

A =0 : Motor Drivers disabled;

A =1 : Motor Drivers enabled;

B =0 : Motor Power not OK;

B =1 : Motor Power OK;

C =0 : Driver Power not OK;

C =1 : Driver Power OK;

D =0 : Electronic Power not OK;

D =1 : Electronic Power OK.

Get temperature command:

[Command] = 0x52

Write -> [0x52]

Read -> [0x52] [Left front Motor Temperature] [Right front Motor Temperature] [Left rear Motor Temperature] [Right rear Motor Temperature] [Left front Motor Driver Temperature] [Right front Motor Driver Temperature][Left rear Motor Driver Temperature] [Right rear Motor Driver Temperature] [Temperature Status] [Send Number] [CheckSum_H] [CheckSum_L]

where:

[Left front Motor Temperature] is the temperature of the left front motor in ºC;

[Right front Motor Temperature] is the temperature of the right front motor in ºC;

[Left rear Motor Temperature] is the temperature of the left rear motor in ºC;

[Right rear Motor Temperature] is the temperature of the right rear motor in ºC;

[Left front Motor Driver Temperature] is the temperature of the Left front Motor Driver in ºC;

[Right front Motor Driver Temperature] is the temperature of the Right front Motor Driver in ºC;

[Left rear Motor Driver Temperature] is the temperature of the Left rear Motor Driver in ºC;

[Right rear Motor Driver Temperature] is the temperature of the Right rear Motor Driver in ºC;

[Temperature Status] is the temperature status [ABCDEFGH]

where:

A=0 : Left front Motor Temperature less that 40ºC;

A=1 : Left front Motor Temperature more that 40ºC;

B=0 : Right front Motor Temperature less that 40ºC;

B=1 : Right front Motor Temperature more that 40ºC;

C=0 : Left rear Motor Temperature less that 40ºC;

C=1 : Left rear Motor Temperature more that 40ºC;

D=0 : Right rear Motor Temperature less that 40ºC;

D=1 : Right rear Motor Temperature more that 40ºC;

E=0 : Left front Motor Driver Temperature less that 40ºC;

E=1 : Left front Motor Driver Temperature more that 40ºC;

F=0 : Right front Motor Driver Temperature less that 40ºC;

F=1 : Right front Motor Driver Temperature more that 40ºC.

G=0 : Left rear Motor Driver Temperature less that 40ºC;

G=1 : Left rear Motor Driver Temperature more that 40ºC;

H=0 : Right rear Motor Driver Temperature less that 40ºC;

H=1 : Right rear Motor Driver Temperature more that 40ºC.

note: The 40ºC value can be changed after a better understanding of the normal operation of the robot.

Set cooling fans command:

[Command] = 0x53

Write -> [0x53][Fan Control]

Read -> [0x53][Send Number] [CheckSum_H] [CheckSum_L]

where:

[Fan Control] allow automatic or manual control of the fans to cool the motors and the drivers. [AXXXBCDE]

[A]=0 manual control of the fans

[A]=1 automatic control of the fans. One or more fans will start to cool a motor or a motor driver when its temperature reaches the 40ºC and stop to work when the temperature is reduced to about 25º. It will not be possible to control the fans manually.

In manual control mode:

[B] = 0 : turn OFF the front motor fans;

[B] = 1 : turn ON the front the motor fans;

[C] = 0 : turn OFF the rear motor fans;

[C] = 1 : turn ON the rear the motor fans;

[D] = 0 : turn OFF the front motor driver fans;

[D] = 1 : turn ON the front motor driver fans;

[E] = 0 : turn OFF the rear motor driver fans;

[E] = 1 : turn ON the rear motor driver fans.

In automatic control the controller will update automatically the [B] to [E] bits. Using 0 when the fans are OFF and 1 when the fans are ON.

note: The 40ºC and 25ºC values can be changed after a better understanding of the normal operation of the robot.

Get cooling fans command:

[Command] = 0x54

Write -> [0x54]

Read -> [0x54][Fan Control][Send Number] [CheckSum_H] [CheckSum_L]

where:

[Fan Control] reads the Fan Control status byte described previously on the “Set cooling fans command”

Set Hardstop timer command:

[Command] = 0x58

Write -> [0x58][Hardstop Timer]

Read -> [0x58][Send Number] [CheckSum_H] [CheckSum_L]

where:

[Hardstop Timer] allow to set the time that the robot will stay in Hardstop Mode. It is possible to select between 0s to 255s.

Get Hardstop Status:

[Command] = 0x59

Write -> [0x59]

Read -> [0x59][Hardstop Status][Send Number] [CheckSum_H] [CheckSum_L]

where:

[Hardstop Status] is the Hardstop status [XXXXXXXA].

where:

A =0 : Hardstop Disable;

A =1 : Hardstop Enable;

Get Firmware version number command:

[Command] = 0x20

Write -> [0x20]

Read -> [0x20] [Firmware information (25bytes)] [Send Number] [CheckSum_H] [CheckSum_L]

where:

[Firmware information] is a set of 25 bytes in ascii with the following information:

“Board2 fw A.BC YYYY/MM/DD” for example “Board2 fw 1.00 2014/02/14”

A is the version and BC the revision number.

A.3. IMU Communication Protocol

Sample IMU Data output

[Command] = 0x40

Write -> [0x40]

Read -> [0x40] [Yaw_High] [Yaw_Low][Pitch_High] [Pitch_Low][Roll_High][Roll_Low][Send Number][CheckSum_H] [CheckSum_L]

The IMU sensor retrieves the information of the Yaw, Pitch and Roll angles in degrees. To obtain the Yaw/Pitch/Roll angles in degrees it is necessary to make the following calculation:

Yaw=(double) [Yaw_High:Yaw_Low]/100,0;

Pitch=(double) [Pitch_High:Pitch_Low]/100,0;

Roll=(double) [Roll_High:Roll_Low]/100,0;

Stream IMU Data output

[Command] = none

Write -> none

Read -> [0x40] [Yaw_High] [Yaw_Low][Pitch_High] [Pitch_Low][Roll_High][Roll_Low][Send Number][CheckSum_H] [CheckSum_L]

The IMU sensor retrieves the information of the Yaw, Pitch and Roll angles in degrees. In streaming mode the IMU will send the angle information with 100Hz updates. To obtain the Yaw/Pitch/Roll angles in degrees it is necessary to make the following calculation:

Yaw=(double) [Yaw_High:Yaw_Low]/100,0;

Pitch=(double) [Pitch_High:Pitch_Low]/100,0;

Roll=(double) [Roll_High:Roll_Low]/100,0;

Disable/Sample/Stream IMU data flow

[Command] = 0x50

Write -> [0x50][Control]

Read -> [0x50] [Send Number] [CheckSum_H] [CheckSum_L]

where:

[Control] =0 : Disable IMU communication;

[Control] =1 : Stream Mode Activated.

[Control] =2 : Sample Mode Activated;

A.4. Interaction Communication Protocol (board3)

Set led mouth control

[Command] = 0x40

Write -> [0x40][32 bytes of 8 LEDs]

Read -> [0x40] [Send Number] [CheckSum_H] [CheckSum_L]

where:

[32 bytes of 8 LEDs] - the LED controller is able to control a matrix of 32 LED columns (32 bytes) by 8 LED rows (8 bits). The mouth is organized in 32 LED columns by 8 LED rows. The first byte will control the 8 leds of the first column. To enable a LED the bit must be changed to ‘1’ and '0' to disable.

Set mouth intensity control

[Command] = 0x41

Write -> [0x41][Intensity]

Read -> [0x41] [Send Number] [CheckSum_H] [CheckSum_L]

where:

[Intensity] = [0-15] is the intensity level for the mouth. The maximum level is 15 and the minimum is 0.

Set RGB LEDs control

[Command] = 0x43

Write -> [0x43][RGB_Device][Red_Intensity][Green_Intensity][Blue_Intensity][time]

Read -> [0x43] [Send Number] [CheckSum_H] [CheckSum_L]

where:

[RGB_Device]=0; Left Eye Leds.

[RGB_Device]=1; Right Eye Leds.

[RGB_Device]=2; Cheeks Leds.

[RGB_Device]=3; Base Right Leds.

[RGB_Device]=4; Base Front Leds.

[RGB_Device]=5; Base Left Leds.

[Red_Intensity] = [0-100] is the intensity percentage of the signal that will drive the red LEDs.

[Green_Intensity] = [0-100] is the intensity percentage of the signal that will drive the green LEDs.

[Blue_Intensity] = [0-100] is the intensity percentage of the signal that will drive the blue LEDs.

[time] = [0-255] is the time that will take to reach the new intensity level. Where [0] is 0 seconds [1] is 0.01s and [255] is 2,55s.

Set head rotation angle control

[Command] = 0x44

Write -> [0x44][Angle]

Read -> [0x44] [Send Number] [CheckSum_H] [CheckSum_L]

where:

[Angle] = [0 – 180] is the final angle between the head and body of the robot. When Angle=0 the head is turned to the right, Angle = 90 the head is centred with the body and with the Angle=180 the head is turned to the left.

Set Head velocity Control

[Command] = 0x45

Write -> [0x45][Velocity]

Read -> [0x45] [Send Number] [CheckSum_H] [CheckSum_L]

where:

[Velocity] = [0 – 100] is the percentage of the maximum velocity that the robot is able to rotate the head.

Enable/Deactivate light beacons

[Command] = 0x46

Write -> [0x46][Enable/Disable]

Read -> [0x46] [Send Number] [CheckSum_H] [CheckSum_L]

where:

[Enable/Disable] Allows the user to activate or deactivate the light beacons installed on the robot head.

Enable/Deactivate Projector

[Command] = 0x47

Write -> [0x47][Enable/Disable]

Read -> [0x47] [Send Number] [CheckSum_H] [CheckSum_L]

where:

[Enable/Disable] Allows the user to activate or deactivate the projector installed on the robot head.

Get head angle position

[Command] = 0x50

Write -> [0x50]

Read -> [0x50][Angle][Send Number] [CheckSum_H] [CheckSum_L]

where:

[Angle] = [0 – 180] is the angle between the head and body of the robot. When Angle=0 the head is turned to the right, Angle = 90 the head is centred with the body and with the Angle=180 the head is turning to the left.

Get Arm Switch's

[Command] = 0x51

Write -> [0x51]

Read -> [0x51][Arm_Switchs][Send Number] [CheckSum_H] [CheckSum_L]

where:

[Arm_Switchs] = [XXXXABCD]

A = Left Arm Top Switch;

B= Left Arm Bottom Switch;

C = Right Arm Top Switch;

D = Right Arm Bottom Switch;

if 0 the switch is not activated;

if 1 the switch is activated.

Get Capacitor Touch sensor data

[Command] = 0x52

Write -> [0x52]

Read -> [0x52][Touch_Sensors_L][Touch_Sensors_H][Send Number] [CheckSum_H] [CheckSum_L]

where:

[Touch_Sensors_L] = [HGFEDCBA]

A = Right shoulder;
B= Right arm;
C = Head;
D = Left arm;
E = Left shoulder;
F = Touch Sensor 5; //not in use
G = Touch Sensor 6; //not in use
H = Touch Sensor 7; //not in use

[Touch_Sensors_H] = [XXXXLKJI]

I = Touch Sensor 8; //not in use
J= Touch Sensor 9; //not in use
K = Touch Sensor 10; //not in use
L = Touch Sensor 11; //not in use

if 0 the touch sensor is not activated;
if 1 the touch sensor is activated.

Get Firmware version number command:

[Command] = 0x20

Write -> [0x20]

Read -> [0x20] [Firmware information (25bytes)] [Send Number] [CheckSum_H] [CheckSum_L]

where:

[Firmware information] is a set of 25 bytes in ascii with the following information:

“Board3 fw A.BC YYYY/MM/DD” for example “Board3 fw 1.00 2014/02/14”

A is the version and BC the revision number.

A.4. Arms (board4)

The SO robots have two independent arms. These arms rotate using two HerkuleX motors that include all the needed hardware and software to make the position control of the arms.

More information about the Herkulex motors can be found in the following link:

http://www.robotshop.com/media/files/pdf/manual-drs-0201.pdf

The motors are powered by the interaction board and use a FTDI USB to RS232 converter to communicate to the each module at the same time. Each Motor has its unique ID. The **Left Arm** will reply to the 0x00 ID and **Right Arm** will reply to the 0x01 ID.

It is possible also to use broadcast 0xFD ID when a command must be sent to all the motors at the same time. This is used, for example, to change the Torque State, from Torque ON, OFF or Break.

There is a software program that is used to change motor setup. The program can be found in the following link:

http://www.dongburobot.com/jsp/board/boardDown.jsp?bseq=6783

The HerkuleX Manager program can be used, for example to, change of the ID, maximum and minimum limits and controller gains. It can also be used to test the motors.

To control the two motor only two functions are needed.

1. Torque Control. There are three possible states for the Torque control:
    - Torque OFF: 0x00 – the motor will be free and will not respond to position commands;
    - Torque ON: 0x60 – the motor will respond to position commands;
    - Break ON: 0x40 – The motor will stay on the same place and will not respond to position commands.

    The Torque Control does not give an acknowledge reply.

```
public void Torque_Control(int ID, byte Torque_Com)
{
        PacketSize = 0x0A;
        CMD = 0x03;
        Data0 = 0x34;
        Data1 = 0x01;
        Data2 = Torque_Com;
        Checksum1 = (byte)((PacketSize ^ ID ^ CMD ^ Data0 ^ Data1 ^ Data2) & 0xFE);
        Checksum2 = (byte)((~Checksum1) & 0xFE);
                serialArms.Write(new byte[] { (byte)header1, (byte)header2, (byte)PacketSize,
(byte)ID, CMD, Checksum1, Checksum2, Data0, Data1, Data2 }, 0, PacketSize);
}
```

where ID:

- 0x00 for the Left Arm;
- 0x01 for the Right Arm;
- 0xFD both Arms;

and Torque_Com:

- Torque OFF: 0x00;
- Torque ON: 0x60;
- Break ON: 0x40.

2. Position Control: Controls the position of the motor shaft. It will receive a new reference position and a delta time that it time that the motor will take from the original position to the new one. The possible position values are represented in the +/- 166,7º, where value 0 = -166,7º, 512 = 0º and 1023 = +166,7º. The Position Control does not give an acknowledge reply.

```
public void Set_Left_Arm_Angle(int Left_Angle, byte Play_Time)
{
        int New_Left_Value;
        byte High_Value, Low_Value;
```

```
            b1=Left_Arm_Minimum_Value;
            m1=(Left_Arm_Maximum_Value-Left_Arm_Minimum_Value)/150.0;

            New_Left_Value = (int)(m1 * (double)Left_Angle + b1);
                    if (New_Left_Value > Left_Arm_Maximum_Value) New_Left_Value =
            Left_Arm_Maximum_Value;
                  else if (New_Left_Value < Left_Arm_Minimum_Value) New_Left_Value =
            Left_Arm_Minimum_Value;
            High_Value=(byte) (New_Left_Value >> 8);
            Low_Value=(byte) (New_Left_Value & 0x00FF);
            I_Jog_TAG_Control_1(0x00, (byte) High_Value, (byte) Low_Value, (byte) Play_Time);

        }

        public void Set_Right_Arm_Angle(int Right_Angle, byte Play_Time)
        {
            int New_Right_Value;
            byte High_Value, Low_Value;
            b1 = Right_Arm_Maximum_Value;
            m1 = (Left_Arm_Minimum_Value - Left_Arm_Maximum_Value) / 150.0;
            New_Right_Value = (int)(m1 * (double)Right_Angle + b1);
                    if (New_Right_Value > Right_Arm_Maximum_Value) New_Right_Value =
            Right_Arm_Maximum_Value;
                  else if (New_Right_Value < Right_Arm_Minimum_Value) New_Right_Value =
            Right_Arm_Minimum_Value;
            High_Value = (byte)(New_Right_Value >> 8);
            Low_Value = (byte)(New_Right_Value & 0x00FF);
            I_Jog_TAG_Control_1(0x01, (byte)High_Value, (byte)Low_Value, (byte)Play_Time);
        }

        private void I_Jog_TAG_Control_1(int ID, byte POS_High, byte POS_Low, int Play_Time)
        {
            PacketSize = 0x0C;
            CMD = 0x05;
            Data0 = POS_Low;
            Data1 = POS_High;
            Data2 = (byte)Led_Color;
            Data3 = (byte)ID;
            Data4 = (byte)(Play_Time);

             Checksum1 = (byte)((PacketSize ^ ID ^ CMD ^ Data0 ^ Data1 ^ Data2 ^ Data3 ^ Data4) &
            0xFE);
            Checksum2 = (byte)((~Checksum1) & 0xFE);
             serialArms.Write(new byte[] { (byte)header1, (byte)header2, (byte)PacketSize, (byte)ID,
            CMD, Checksum1, Checksum2, Data0, Data1, Data2, Data3, Data4 }, 0, PacketSize);
        }
```

where:

```
   Left_Arm_Minimum_Value  = 100;
   Left_Arm_Maximum_Value  = 900;
   Right_Arm_Minimum_Value = 100;
   Right_Arm_Maximum_Value = 900;
```

To read information from two motor the following functions can be implemented.

The principal function is the Read_Info_Status. This function will call the desired function that the users wants to receive, from one of the motors and it will return an int32 value.

```
        private Int32 Read_Info_STATUS (int ID, byte Memory, byte N_bytes)
         {
             byte[] buffer = new byte[100];
             Int32 Status;

             PacketSize = 0x09;
             CMD = 0x04;
             Data0 = Memory;
             Data1 = N_bytes;

             Checksum1 = (byte)((PacketSize ^ ID ^ CMD ^ Data0 ^ Data1) & 0xFE);
             Checksum2 = (byte)((~Checksum1) & 0xFE);

             serialArms.Write(new byte[] { (byte)header1, (byte)header2,
                                          (byte)PacketSize, (byte)ID,
                                          CMD, Checksum1, Checksum2,
                                          Data0, Data1 }, 0, PacketSize);

             int count = 0;

             if ( N_bytes == 1 )
                    while (serialArms.BytesToRead < 12 && count <6000)
                    count++;

             else if ( N_bytes == 2 )
                    while (serialArms.BytesToRead < 13 && count <6000) count++;


             if ( N_bytes == 1)
             {
                 if (serialArms.BytesToRead == 12)
                 {
                     serialArms.Read(buffer, 0, serialArms.BytesToRead);
                     buffer[0] = buffer[0];
                     Status = (Int32)buffer[9]  * 0x00010000 +
                              (Int32)buffer[10]  * 0x00000100 +
                              (Int32)buffer[11]* 0x00000001;
                     return (Status);
                 }
             }
             else if ( N_bytes == 2)
             {
                 if (serialArms.BytesToRead == 13)
                 {
                     serialArms.Read(buffer, 0, serialArms.BytesToRead);
                     buffer[0] = buffer[0];
                     Status = (Int32)buffer[10]  * 0x01000000 +
                              (Int32)buffer[9]   * 0x00010000 +
                              (Int32)buffer[11]  * 0x00000100 +
                              (Int32)buffer[12]* 0x00000001;
                     return (Status);
                 }
             }
             return (-1);
         }
```

where:

ID:

- 0x00 for the Right Arm;

- 0x01 for the Left Arm.

Memory and **N_Bytes:**

- 5 – to read the Max_Temperature_Value – 1 Byte;
- 52 – to read the Torque_Control_Value – 1 Byte;
- 54 – to read the Voltage_Value – 1 Byte;
- 55 – to read the Temperature_Value – 1 Byte;
- 60 – to read the Absolute_Position_Value - 2 Byte.

Using the previous function it it possible to create the following functions to read the data from the motor.

1. Read actual position of the motor

```
public int Read_Position(int ID)
  {
      Int32 Position_Int32;
      Int16 Position;

      Position_Int32 = Read_Info_STATUS(ID, Absolute_Position_Value, 2);
       //The position will return a two bytes
      if (Position_Int32 == -1) return (-1);

      Position = (Int16)( Position_Int32 >> 16 );

      return ((int)Position);

  }
```

where the **Position** value is a value between 0 and 1024.

2. Read actual temperature of the motor

```
 public int Read_Temperature(int ID)
  {
      Int32 Temperature_Int32;
      Int16 Temperature;

      Temperature_Int32 = Read_Info_STATUS(ID, Temperature_Value, 1);
       //Return 1 bytes
      if (Temperature_Int32 == -1) return (-1);

      Temperature = (Int16)((double)((Int16)(Temperature_Int32 >> 16)) *
                    0.7105 - 79.47);

      return ((int)Temperature);
  }
```

where **Temperature** is the temperature value between 0-100ºC

3. Read actual Torque Control

```
  public int Read_Torque_Control(int ID)
  {
      Int32 Torque_Control_Int32;
      Int16 Torque_Control;
```

```
        Torque_Control_Int32 = Read_Info_STATUS(ID,
                               Torque_Control_Value, 1);

        if (Torque_Control_Int32 == -1) return (-1);

        Torque_Control = (Int16)(Torque_Control_Int32 >> 16);

        return (Torque_Control);
    }
```

where **torque control**

- Torque OFF: 0x00;
- Torque ON: 0x60;
- Break ON: 0x40.

4. Read actual Status Error

The HerkuleX motors have an error flag that show, in case of error, the problem with the motor.

It is possible to identify the following errors:

- Vin error (the powering voltage is to low or high);
- Position Limit error (for some reason the actual position is outside the maximum and minimum defined limits);
- Temperature Error (if the temperature is higher that 85ºC the flag is activated);
- Packet Error (there was a communication error);
- Overload Error (there was a force higher that the maximum defined that was applied to the motor);
- Driver error (there is some problem with the motor driver);
- EEP error (problem in the Controller EEPROM).

```
public int Read_Status_Error(int ID)
{
    Int32 Status_Error_Int32;
    Int16 Status_Error;

    Status_Error_Int32 = Read_Info_STATUS(ID, Torque_Control_Value, 1);
    //1 byte
    // All commands send the r(Status_Error) +r(Status_Detail)
    if (Status_Error_Int32 == -1) return (-1);

    Status_Error = (Int16)(Status_Error_Int32 & 0xFFFF);

    return ((int)Status_Error);
}
```

where **Status Error** is the [Status_error:Status_Detail]

Using the HerkuleX manager it is possible to define if an error will put the torque to OFF or not. By default all the error messages will turn OFF the torque.

5. Clear the Status Error

After having an error the user must clean the status by sending a STATUS clean message. If, for example, the temperature error was detected, the temperature error bit of the error STATUS will be 1. After some time the temperature will drop but the motor will continue to be at the Torque OFF state, to take it out of it, a Clean STATUS must be sent. And then it will be possible to change the torque to ON and control the motor.

```
public void Clear_Status_Error(int ID)
{
    PacketSize = 0x0B;
    CMD = 0x03;
    Data0 = 0x30;
    Data1 = 02;
    Data2 = 00;
    Data3 = 00;

    Checksum1 = (byte)((PacketSize ^ ID ^ CMD ^ Data0 ^ Data1 ^ Data2 ^
    Data3) & 0xFE);
    Checksum2 = (byte)((~Checksum1) & 0xFE);


     serialArms.Write(new byte[] { (byte)header1, (byte)header2,
                    (byte)PacketSize, (byte)ID, CMD,
                    Checksum1, Checksum2, Data0,
                    Data1, Data2, Data3}, 0, PacketSize);
}
```

The Clear STATUS Error does not give an acknowledge reply.

To use other commands refer to the manual.

Annex B. Robot Platform Base CAD Drawings

Figs. 72 to 76 depict the platform mechanics and its measures.

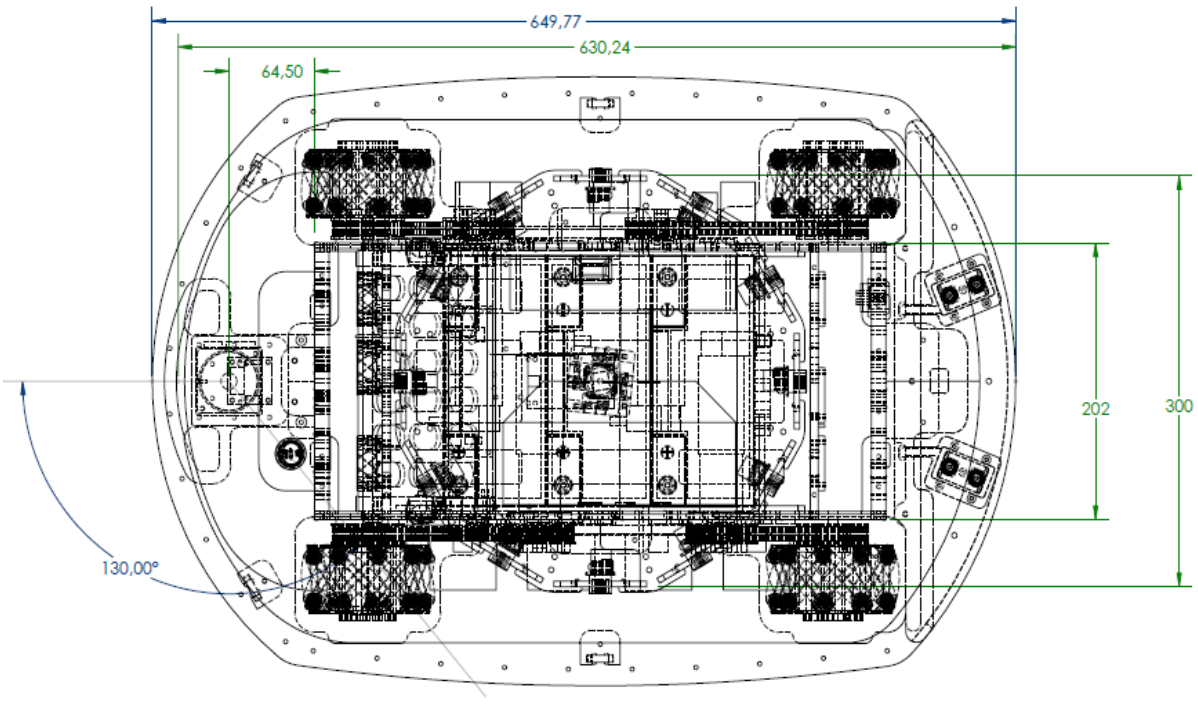

Figure 72: Robot Platform. Top view

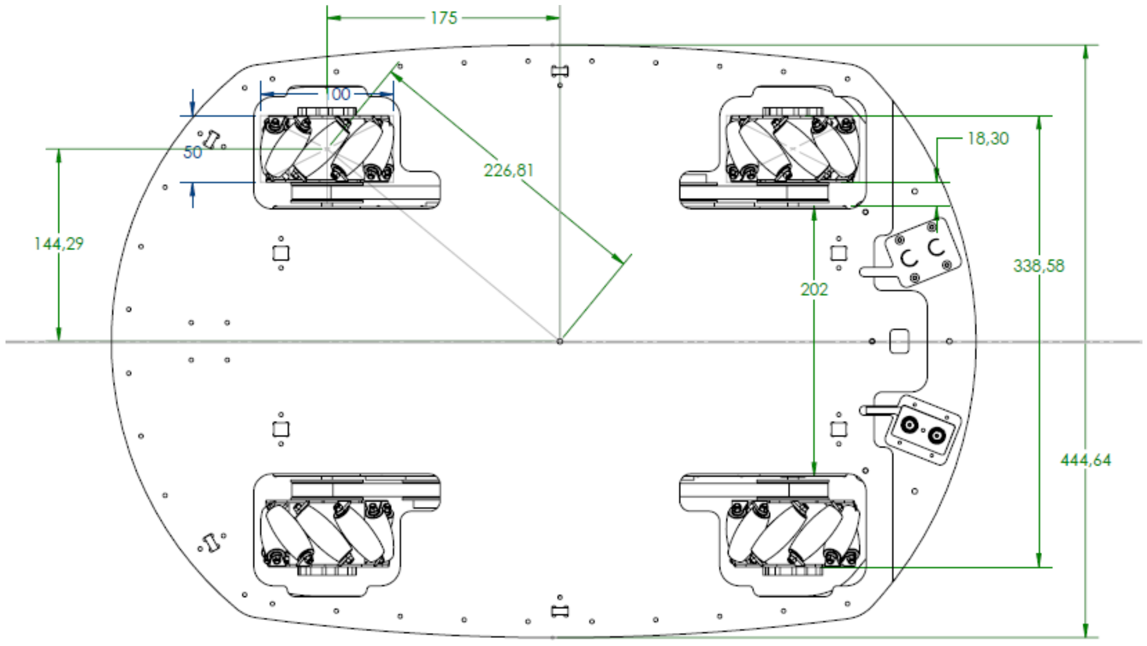

Figure 73: Robot Platform. Bottom view.

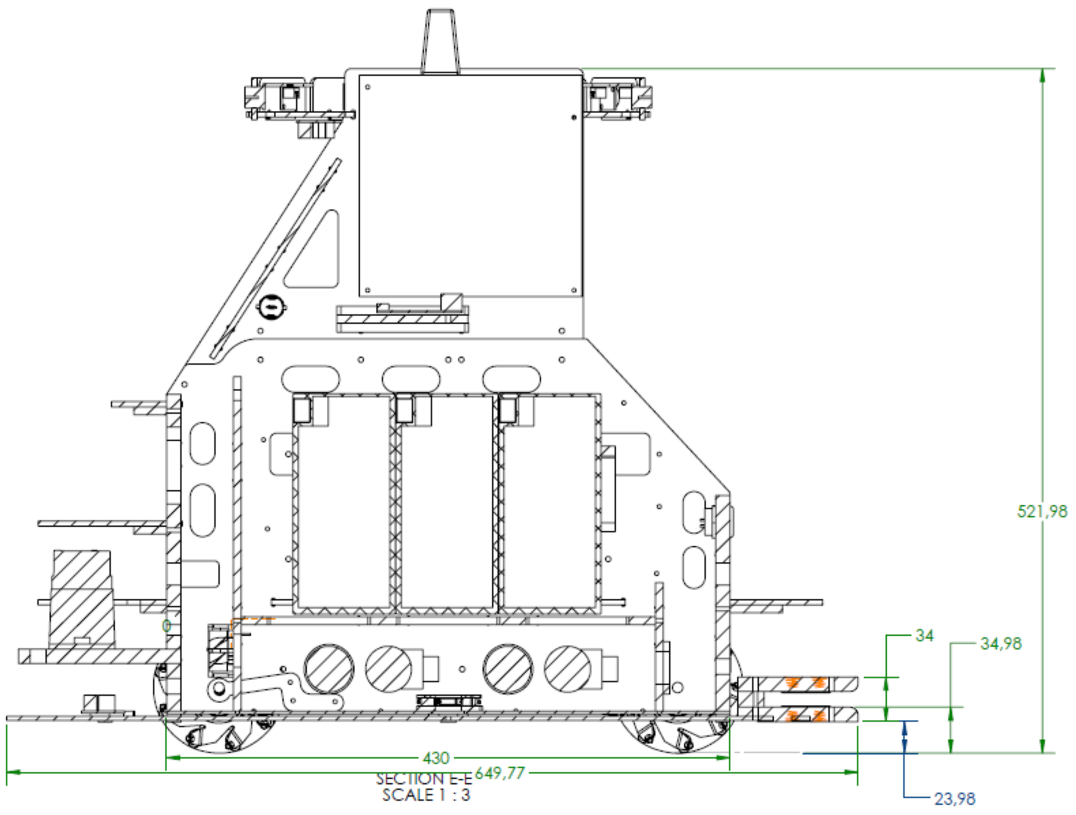

Figure 74: Robot Platform. Left view.

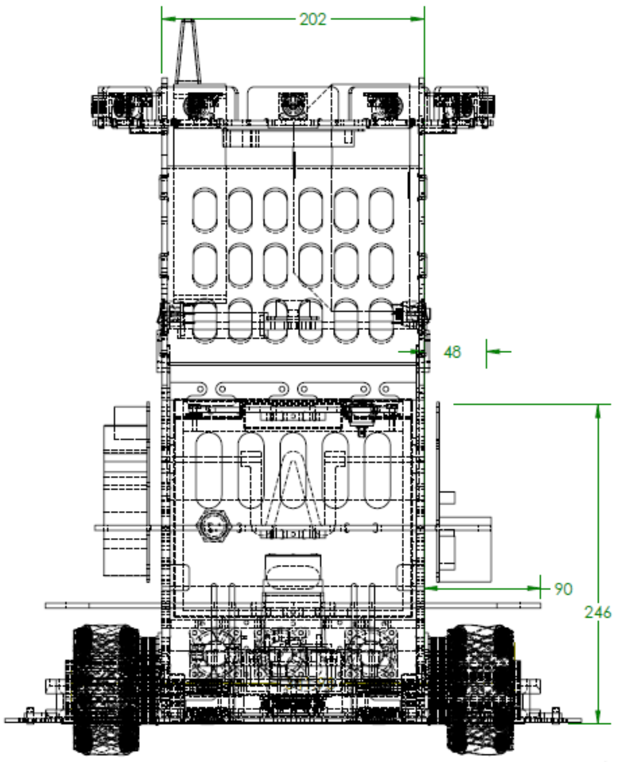

Figure 75: Robot Platform. Front view.

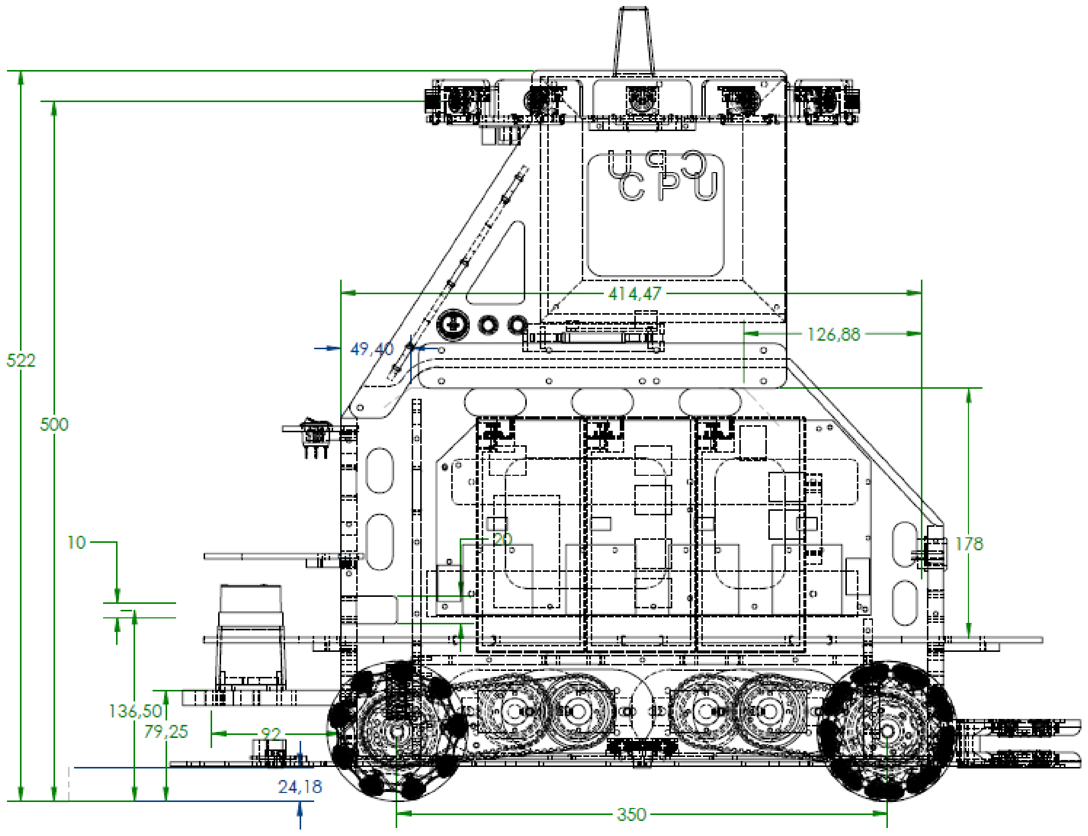

Figure 76: Robot Platform. Pulley system view.

Annex C. Robot CAD Drawings

Figures 77 to 81 depict the MOnarCH robot mechanics and its measures.

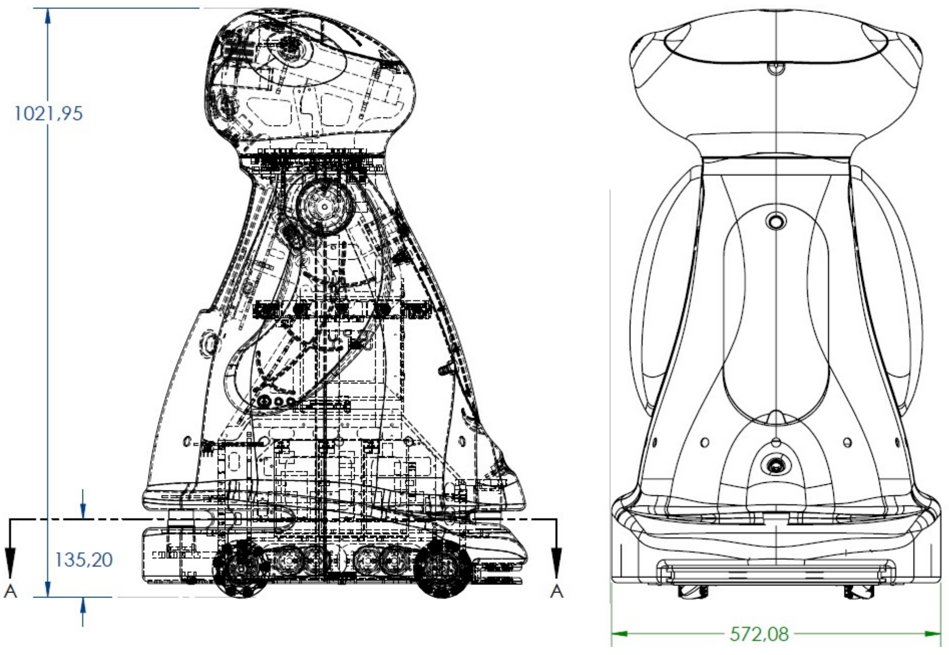

Figure 77: robot height and width

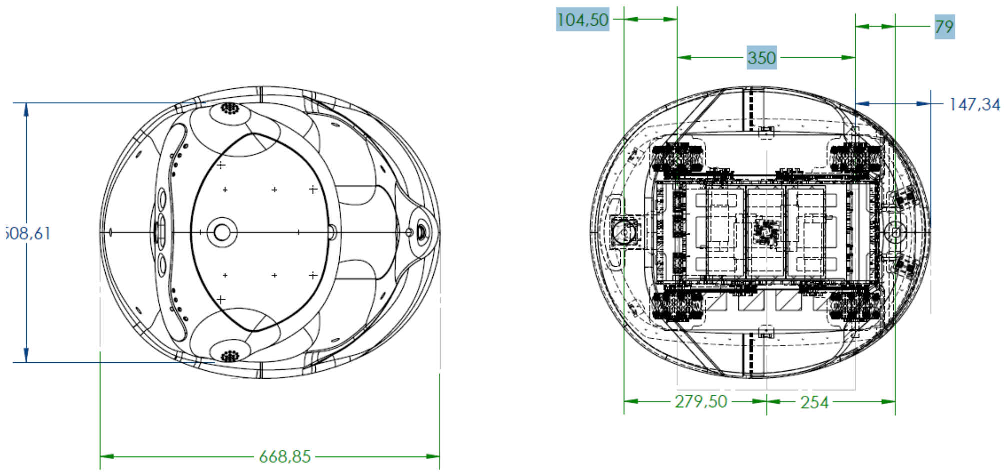

Figure 78: robot depth and laser position

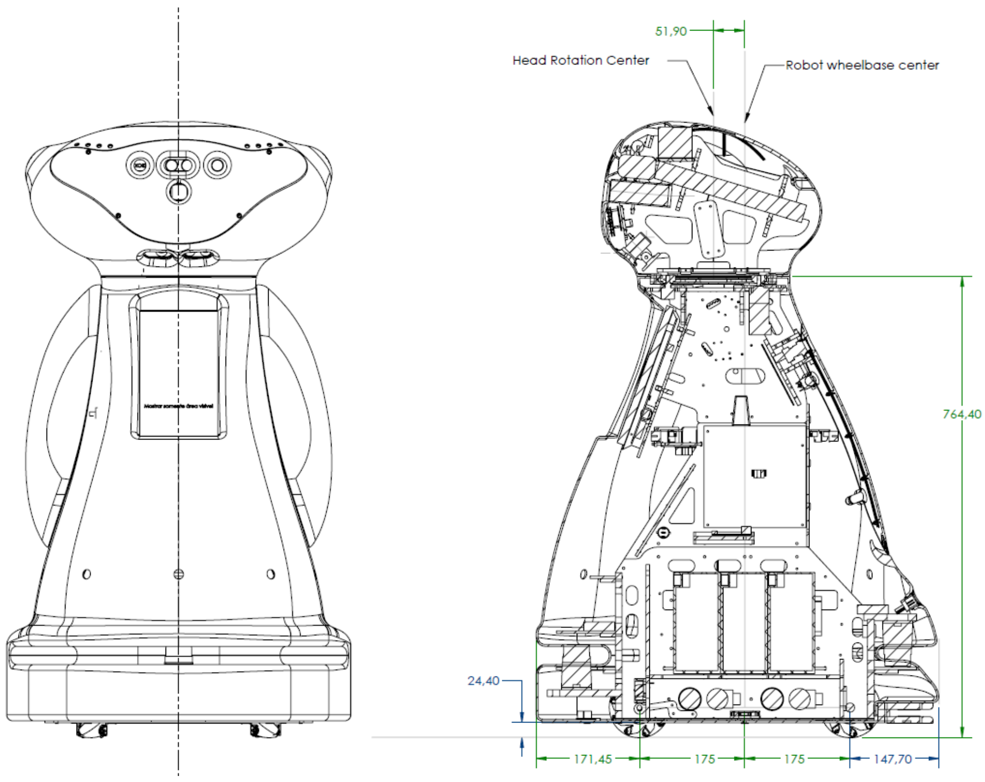

Figure 79: distance between the centre of rotation of the robot and centre of rotation of the head

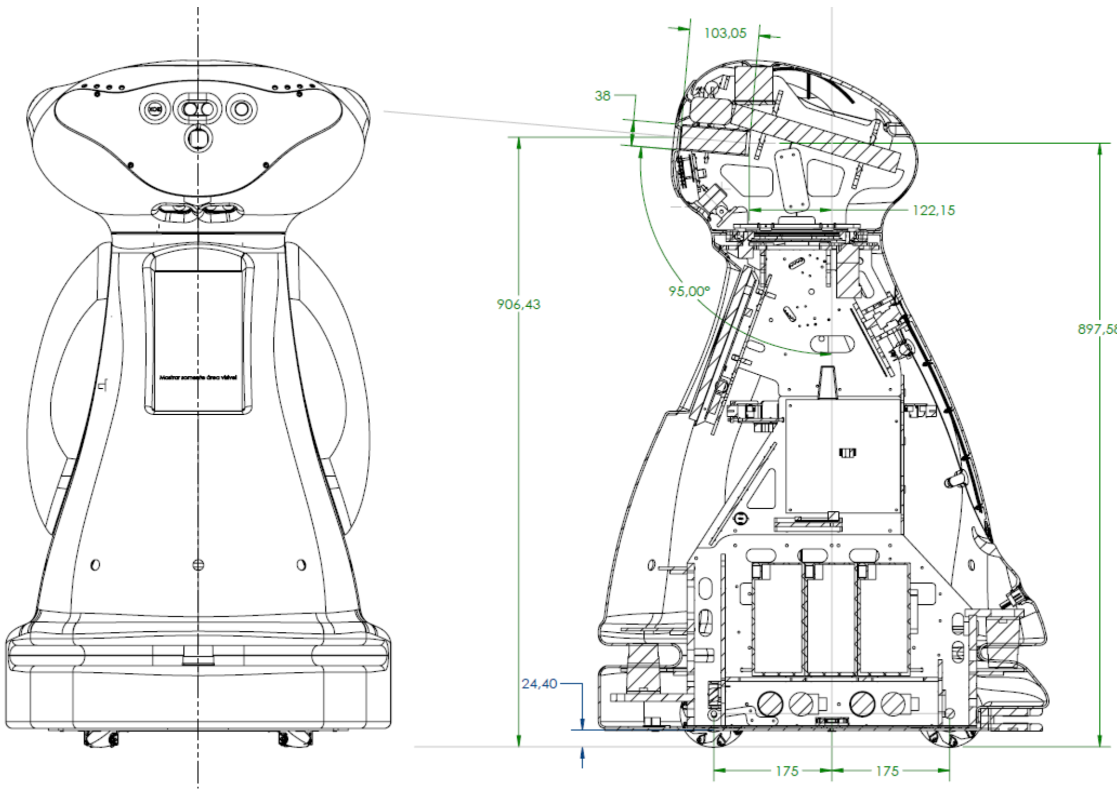

Figure 80 : Position of the Projector from the centre of the robot

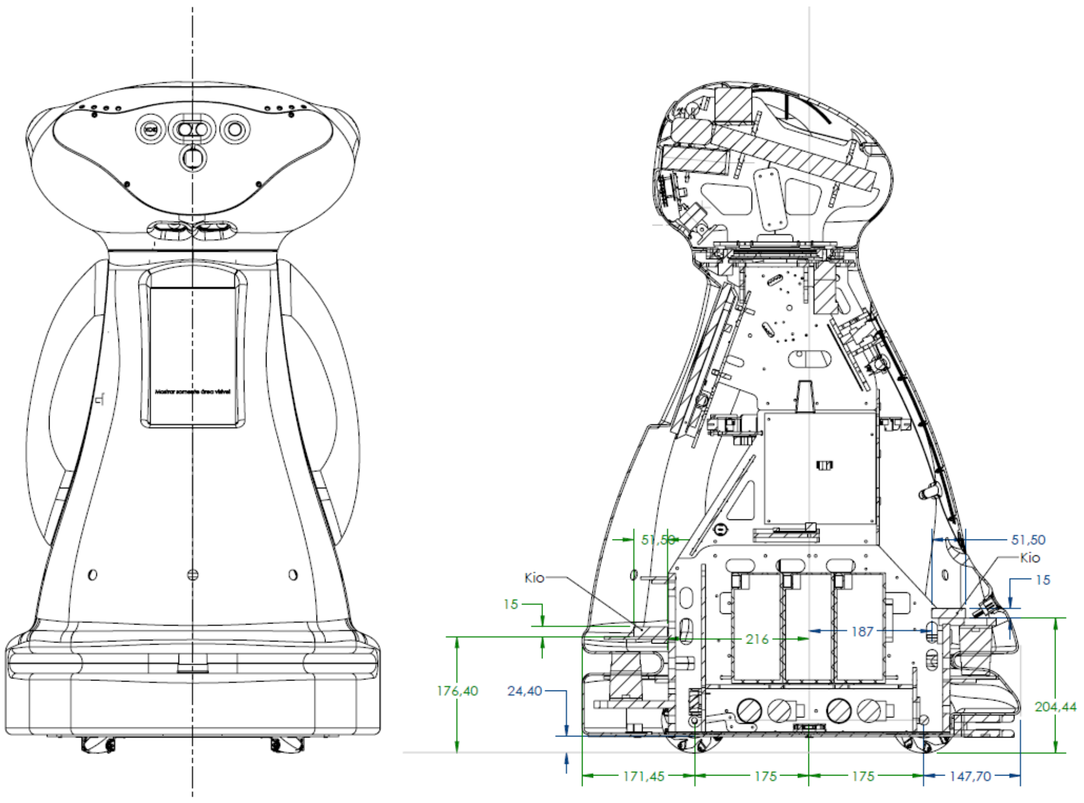

Figure 81 : Position of the UWB sensors from the centre of the robot

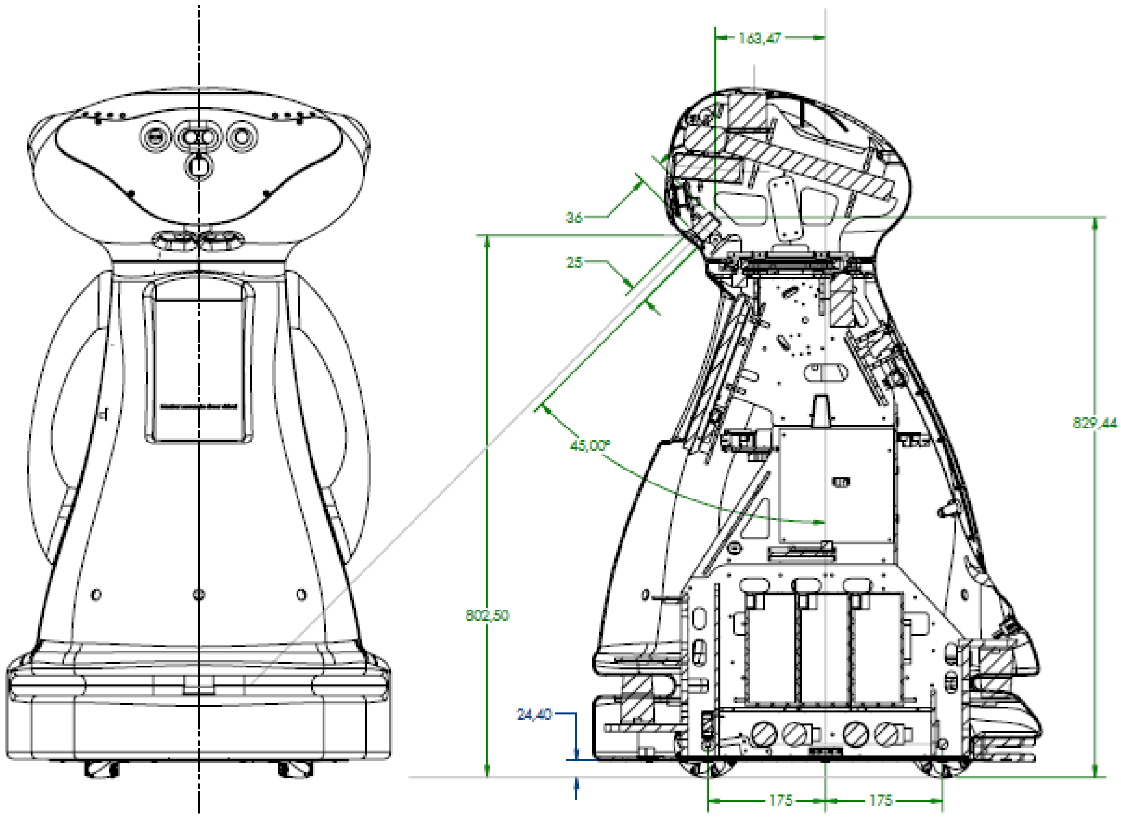

Figure 82 : Position of the Stargazer from the centre of the robot